Détection d'erreur et détection d'intrusion

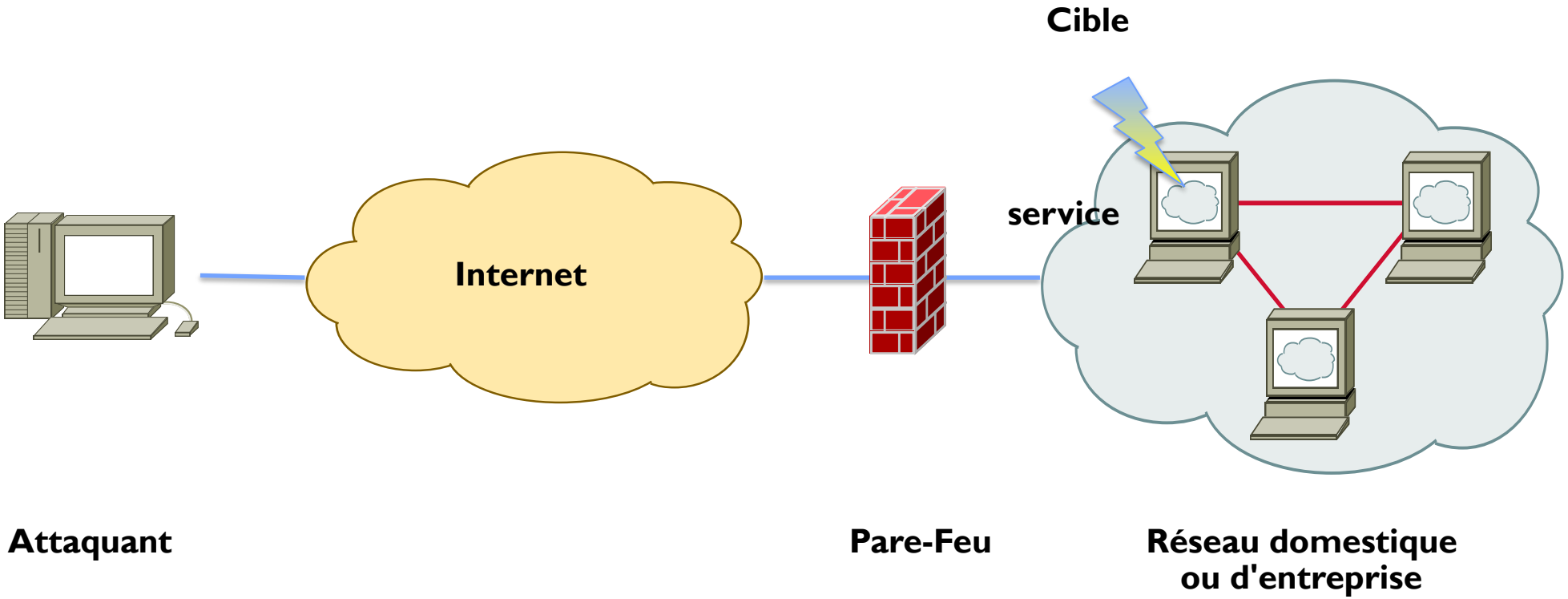
Eric Totel

Eric.Totel@rennes.supelec.fr

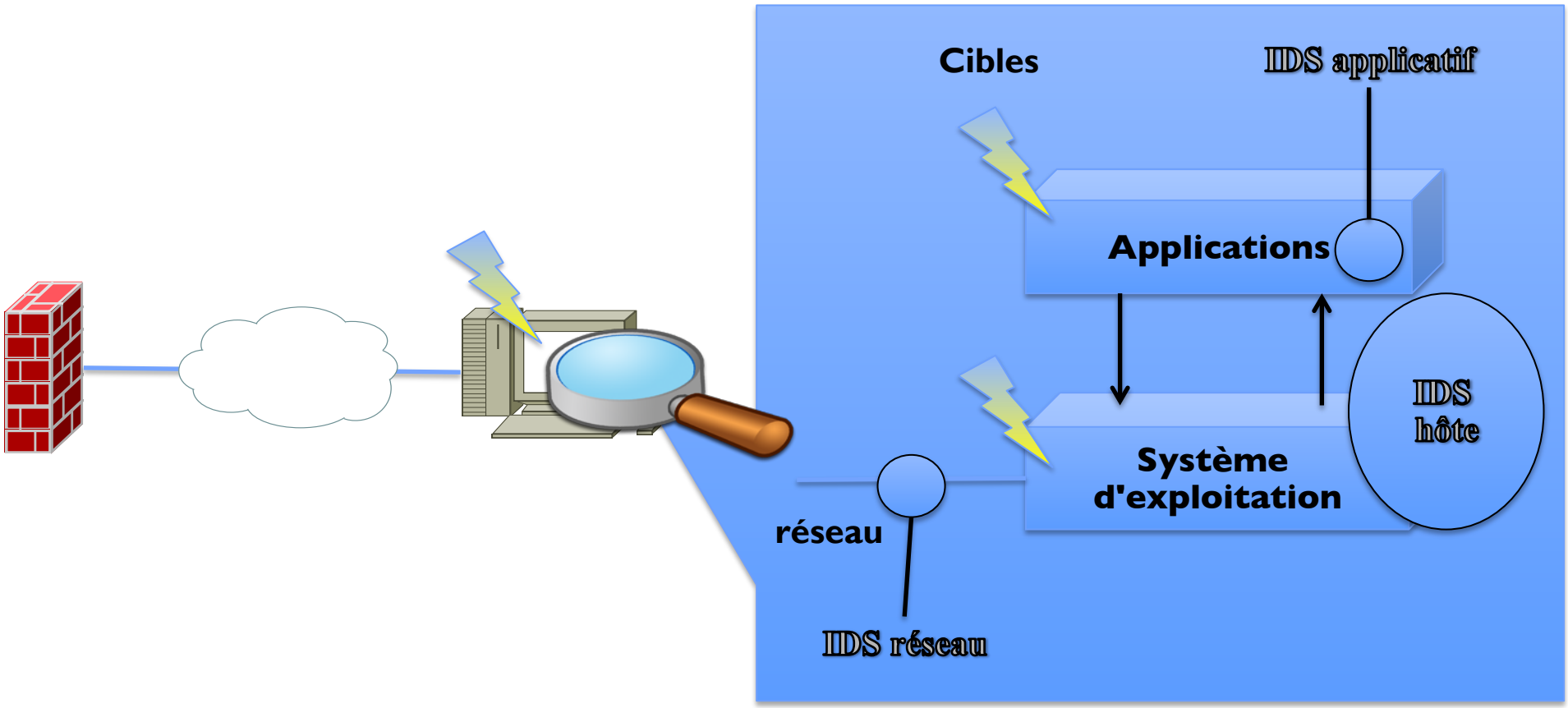
2013

Supélec - Campus de Rennes - France

Positionnement du problème



Sources de données et détection d'intrusion



Comment détecter une intrusion ?

Modèles de détection

- **Approche par signature**
  - Attaques connues, base de signatures
- **Approche comportementale**
  - Déviation par rapport à un comportement normal de référence
  - Tous nos travaux suivent cette approche

Détection comportementale

Modèles de comportements

- Aspects non fonctionnels
  - Règles de sécurité (permettant d'assurer des propriétés de confidentialité ou d'intégrité)
  - Violation éventuelle d'une politique de sécurité

- Aspects fonctionnels
  - Spécification apprise ou connue du service
  - Réalisation de la fonction attendue du service ?
  - Sûreté de fonctionnement : détection d'erreur

Plan de la présentation

1. **Diversification fonctionnelle**
   - **IDS réseau**
   - **IDS Hôte**

2. **Contrôles de vraisemblance**
   - **IDS applicatifs par détection de violation d'invariants (sans apprentissage)**

Plan diversification fonctionnelle

- Introduction aux travaux
  - Diversification fonctionnelle
  - Diversification de COTS
- Détection en « boite noire »
  - Architecture de détection d'intrusion
  - Masquage de différences de spécification
  - Résultats expérimentaux
- Détection en « boite grise »
  - Notion de graphes de flux d'information
  - Mesure de similarité de graphes
- Conclusion

Supélec

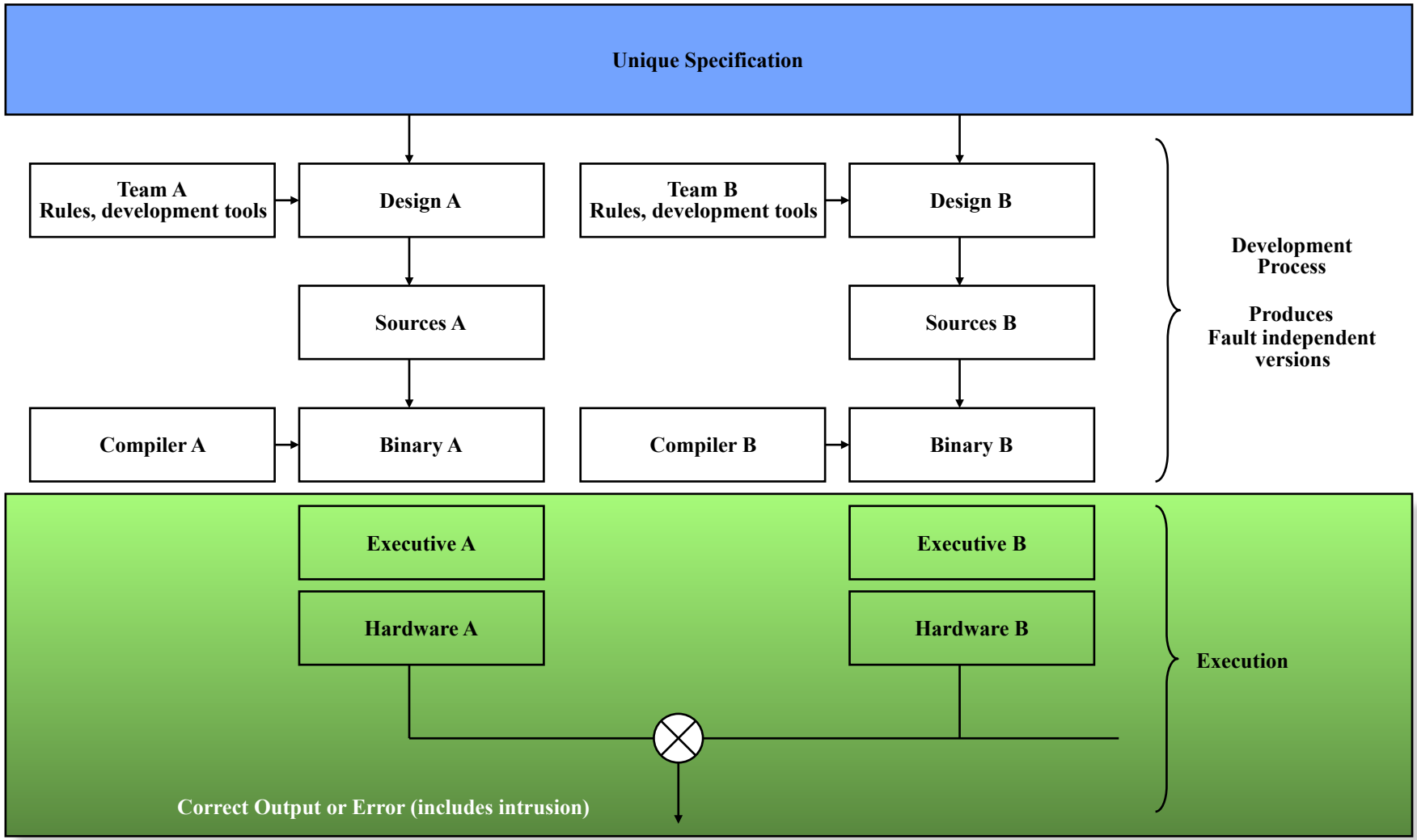
Détection d'intrusion

- **Méthodes**
  - Détection par scénarios
    - Nécessite une connaissance précise des attaques
  - Détection comportementale

- **Détection comportementale**
  - Construction d'un modèle de référence
    - Manuellement ou par apprentissage
  - Comparaison du comportement du système vis-à-vis du modèle
    - Modèle incorrect ou incomplet ⇒ faux négatif ou faux positifs

- **Diversification de COTS:**
  - Comment s'affranchir de la construction *explicite* du modèle

Diversification fonctionnelle: Programmation N-Versions



Bilan sur la diversification fonctionnelle

Hypothèse

- La diversification de la conception et du développement permet la décorrélation des fautes

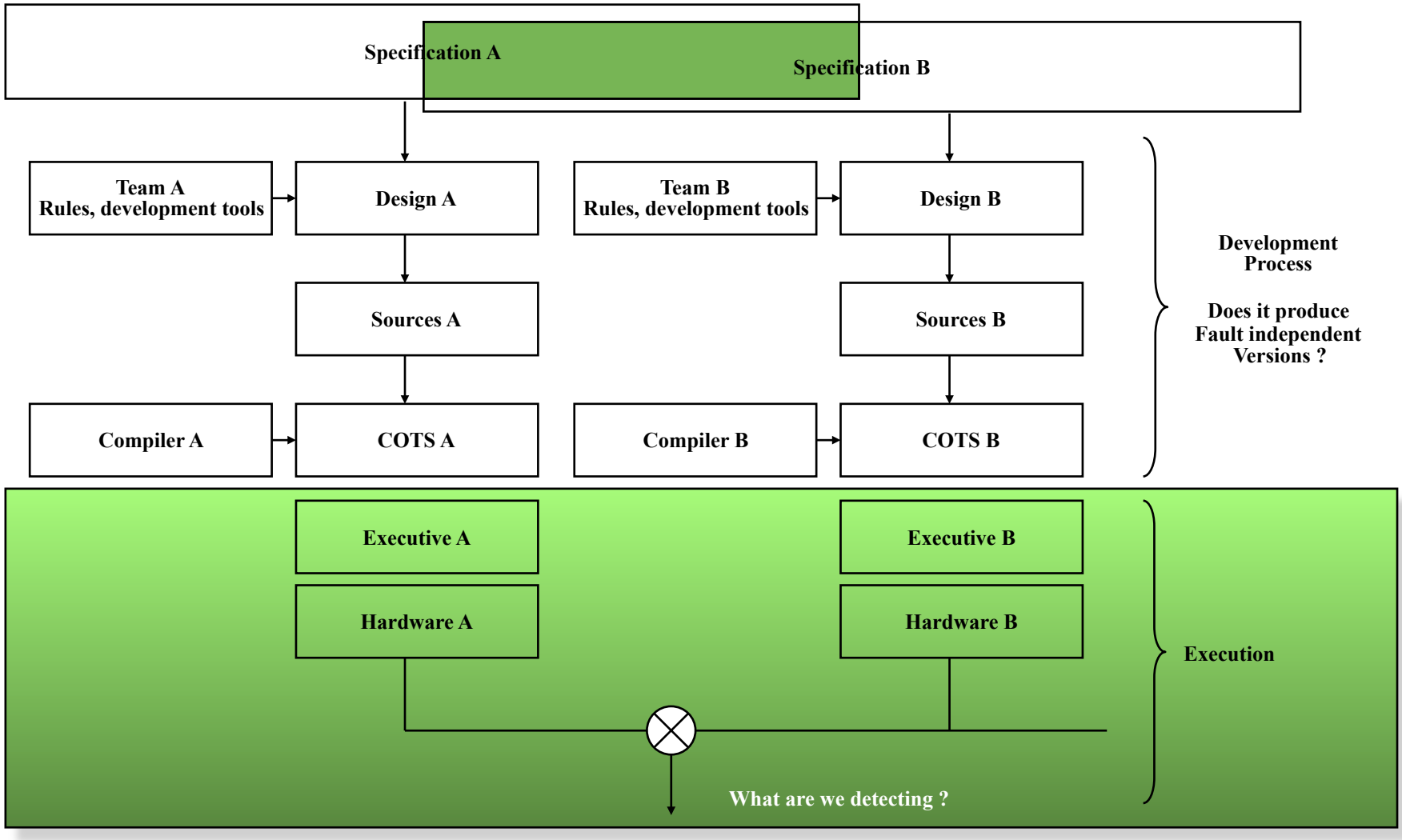
Avantages

- Détection de tout type d'erreurs logicielles qui se propagent jusqu'en sortie du logiciel
  - Différence en sortie est la conséquence de l'activation des fautes
- Chaque logiciel est en fait un modèle de comportement
  - Complet
  - Correct s'il est fidèle à sa spécification
- Tolérance aux fautes:
  - f+2 versions permettent de tolérer f fautes

Inconvénients

- Comment vérifier l'indépendance des fautes ?
  - Par l'expérience
- Coût de développement des N versions
  - Acceptable pour des logiciels très critiques (avionique par exemple)

Diversification de COTS



Bilan sur la diversification de COTS

Avantages

- Coût très faible (logiciels disponibles)

Inconvénients

- Aucune information (ou presque) sur les spécifications
  - On ne connaît souvent que la spécification commune (protocole)
- Comment vérifier l'indépendance des fautes ?
  - Par l'expérience
- Les sorties sont-elles suffisamment bien définies pour être comparables ?
- Que signifie une différence dans les sorties ?

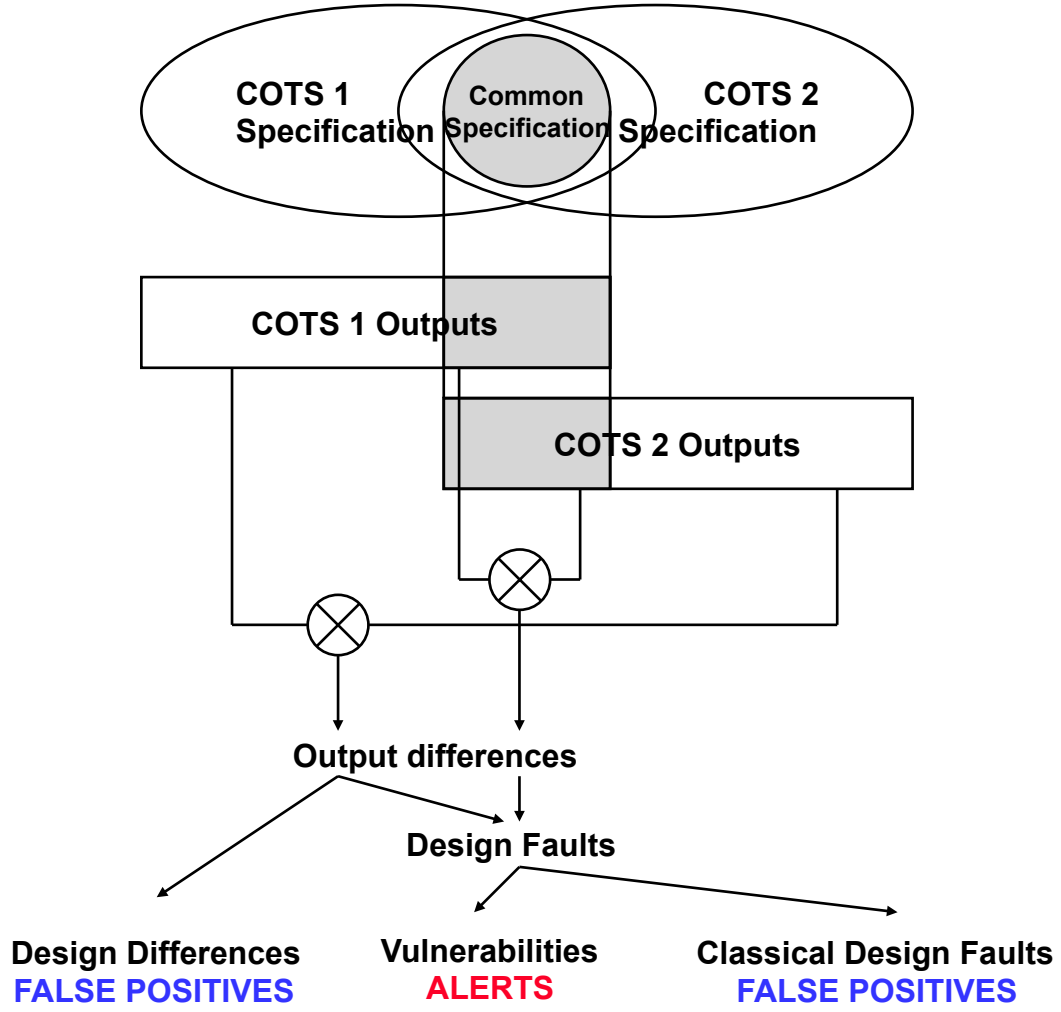
De la détection d'erreurs à la détection d'intrusions

- **Définition de l'intrusion**
  - Une faute qui résulte de la violation de la politique de sécurité du système en terme de
    - Confidentialité
    - Intégrité
    - ou Disponibilité

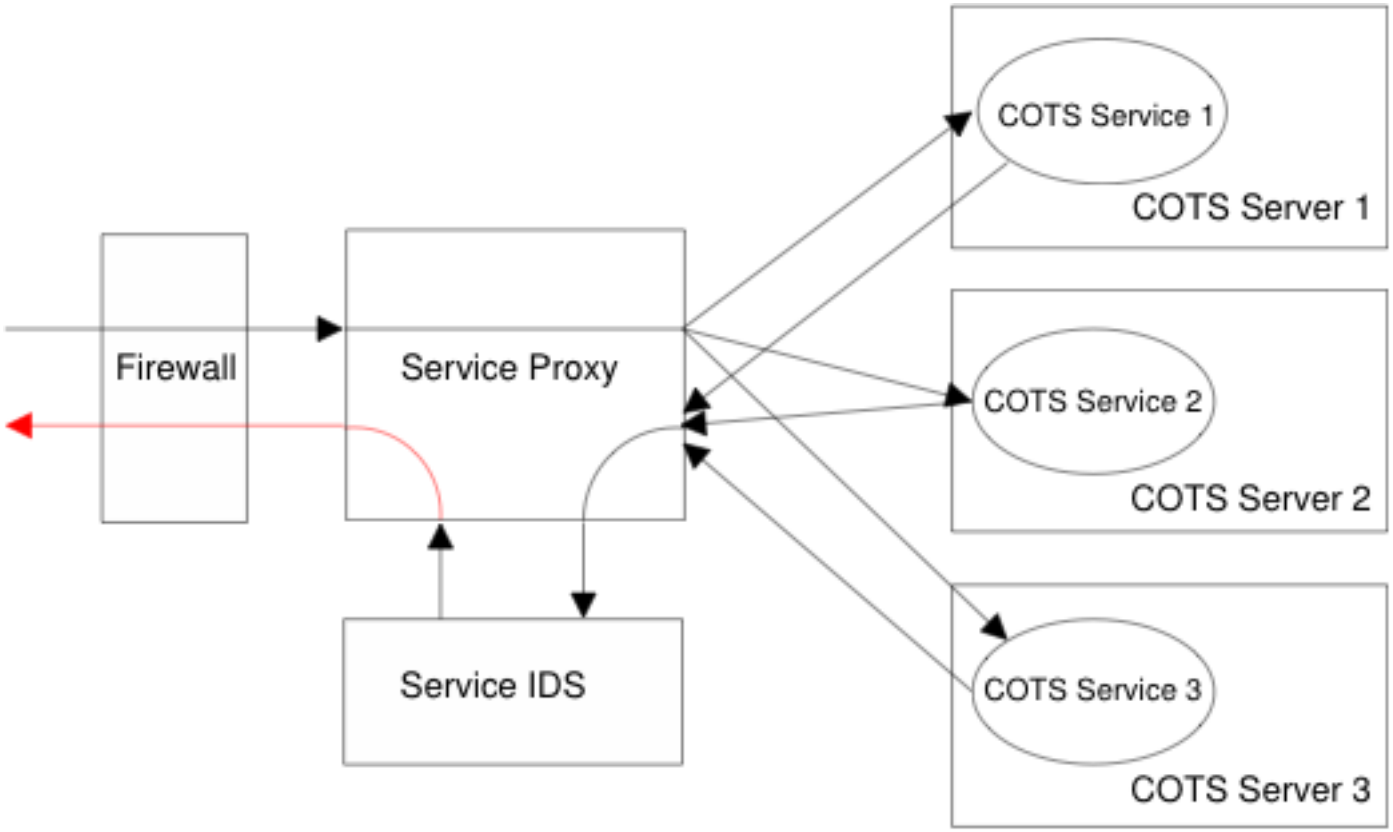
Donc…

- **La diversification fonctionnelle permet la détection**
  - D'erreurs « classiques » (pas de violation de la politique de sécurité)
  - D'erreurs symptomatiques d'une intrusion
- **Alors que la diversification de COTS provoquera la détection**
  - De sorties dues à des différences de spécification
  - D'erreurs « classiques » (pas de violation de la politique de sécurité)
  - D'erreurs symptomatiques d'une intrusion

Comparaison de COTS: que détecte-t'on ?



Architecture de détection d'intrusion



Application à des serveurs Web

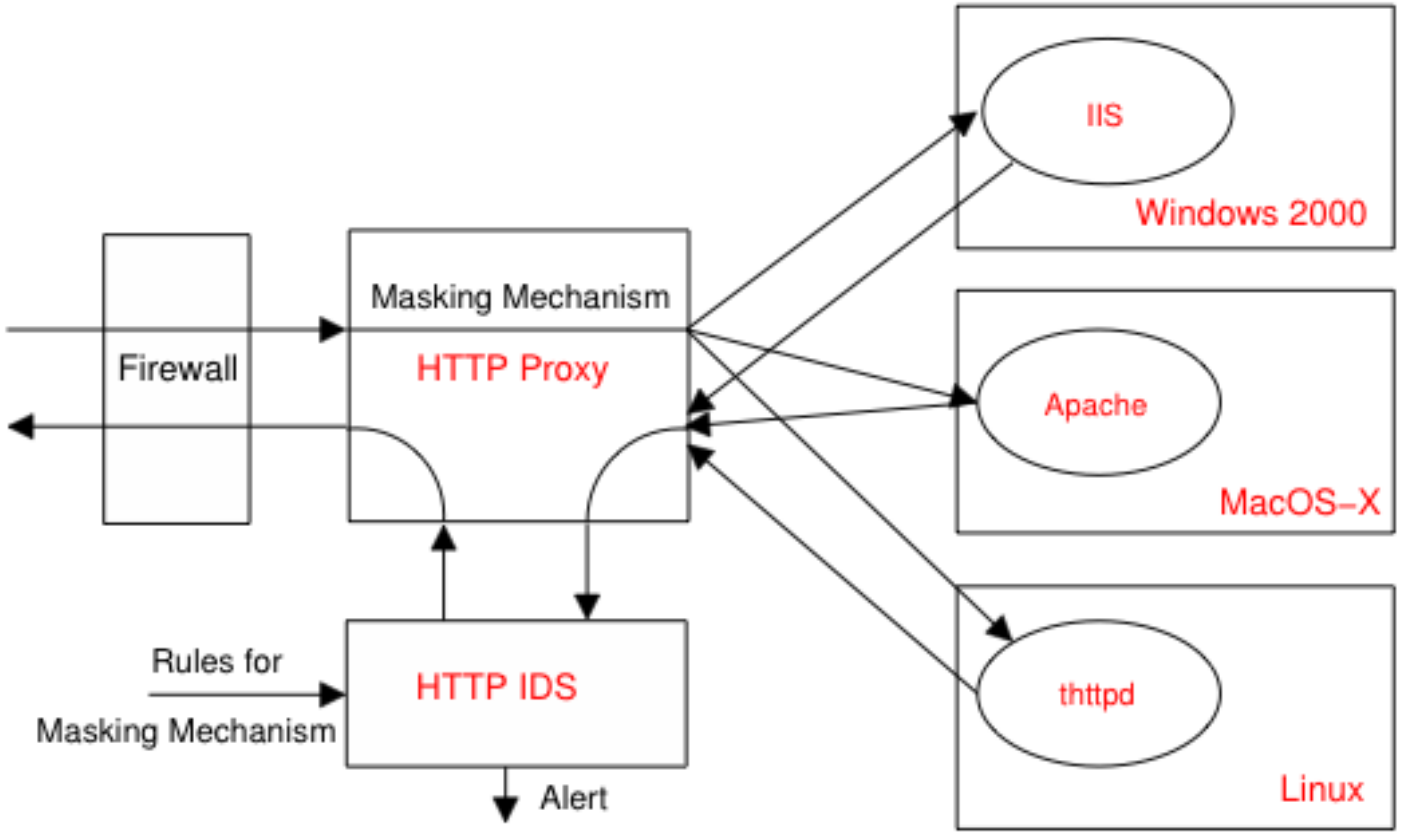
Quelles sorties peut-on/doit-on comparer ?

1. **Approche « boite noire »**
   - **On compare les sorties des serveurs**
   - **Un protocole commun: ici HTTP, donc il existe une spécification commune**
2. **Approche « boite grise »**
   - **Utilisation de la connaissance de ce qui se passe sur le serveur: appels systèmes**
   - **On compare ce qui se passe sur les différents serveurs au niveau applicatif**

Vérification de la décorrélation des intrusions

- **A l'aide des bases de vulnérabilités connues**
  - **Sur les systèmes**
  - **Sur l'application Web**

Application à des serveurs Web



Comparaison en boite noire

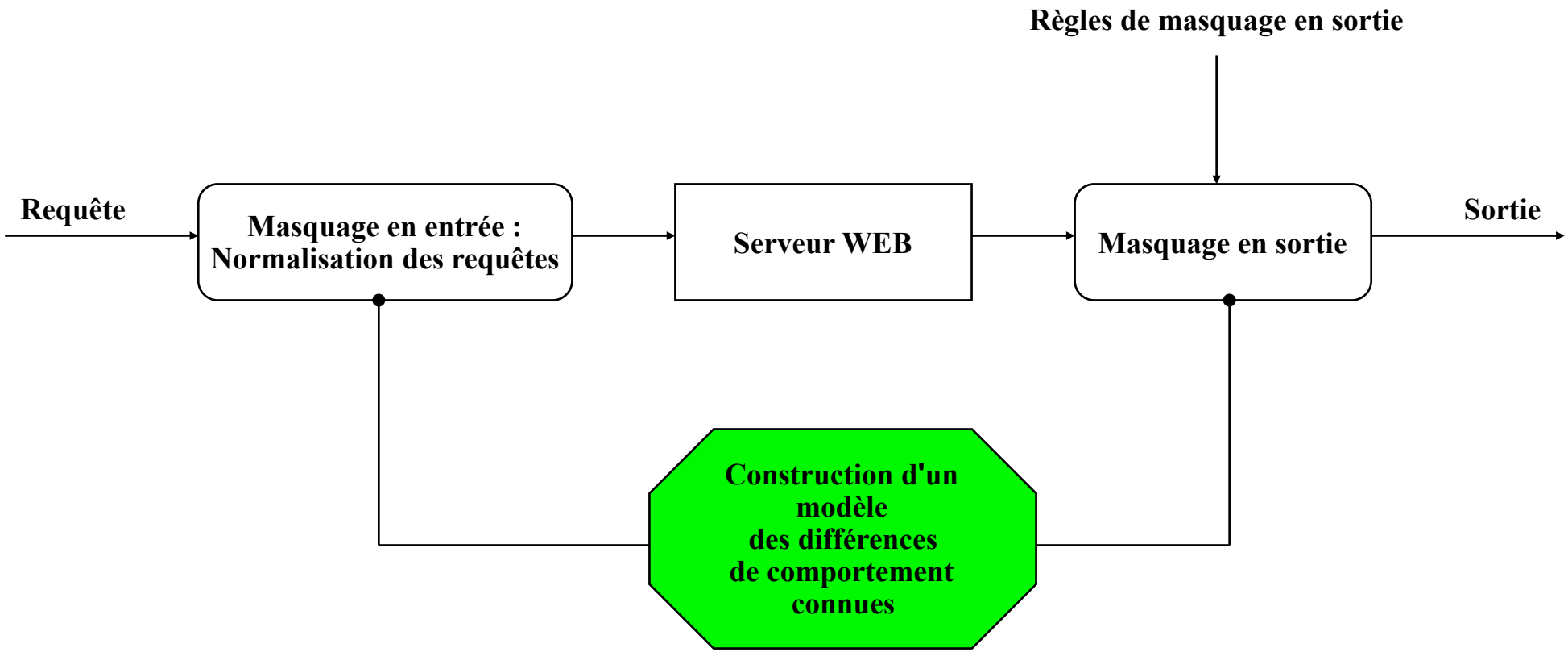
On ne s'interesse qu'aux messages HTTP sortants

- Code de statut, Entêtes HTTP, Corps de la réponse (body HTTP)

Algorithme de détection

- Détection de services non disponibles par un chien de garde (watchdog timer)
- Masquage de différences dues à des différences en sortie ou des fautes de conception «classiques» (i.e., pas de violation de la politique de sécurité)
- Comparaisons successives et vote majoritaire sur:
  - Code de statut
  - Entête HTTP
  - Corps de la réponse
- Une alerte est émise pour chaque serveur qui n'est pas dans la majorité

Masquage de différences



Masquage de différences en sorties

Règle de masquage

- Met en relation
  - Une classe de requêtes
  - Une classe de sortie pour chaque COTS
- Ne doit pas engendrer de faux négatifs
  - Très difficile à démontrer

```
<request id="0">
<regexpURI>^.*[^/]$</regexpURI>
<regexpMethod>^.*$</regexpMethod>
```

```
<serverBehaviour>
        <server>
        <name>apache</name>
        <httpcode>301</httpcode>
        <contentType>text/html</contentType>
        </server>
        <server>
        <name>iis</name>
        <name>thttpd</name>
        <httpcode>302</httpcode>
        <contentType>text/html</contentType>
        </server>
</serverBehaviour>

<compareContent>no</compareContent>
<response>iis</response>
<warn>no</warn>
<alert>no</alert>
</request>
```

Fiabilité de la détection

Environnement de test

- IIS 5.0 / Windows 2000
- Apache 1.3.29 / MacOSX
- buggyHTTP / Linux

Attaques contre la confidentialité, l'intégrité et la disponibilité

- 3 types de vulnérabilités exploitées contre Buggy HTTP
- Vulnérabilité CAN-2001-0925 contre Apache
- Vulnérabilité CVE-2000-0884 contre IIS

Détection

- Toutes les attaques testées ont été détectées
- Pas de faux négatifs

Exactitude de la détection

Environnement de test

- **IIS 5.0 / Windows 2000**
- **Apache 1.3.29 / MacOSX**
- **thttpd / Linux**

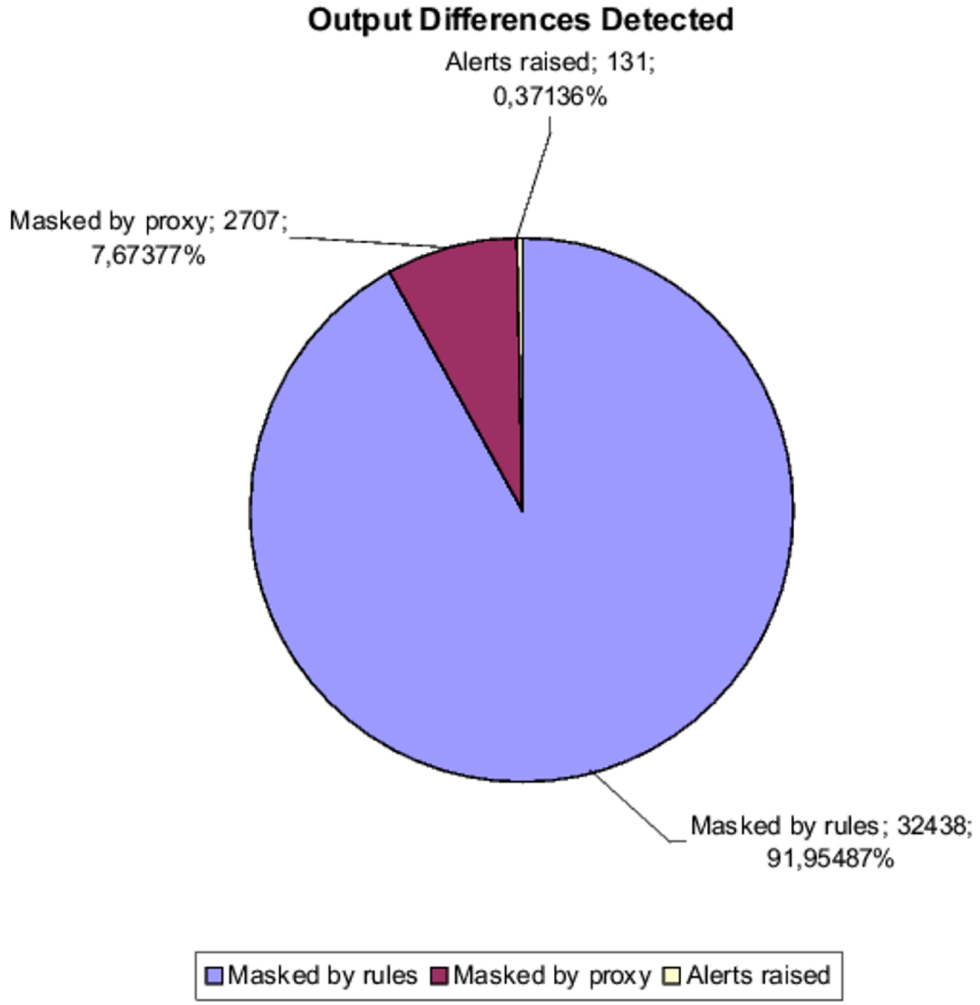
Description des tests

- **Copie des pages du serveur Web de Supelec**
- **Trafic composé de 800 000 requêtes (un mois de log du serveur)**

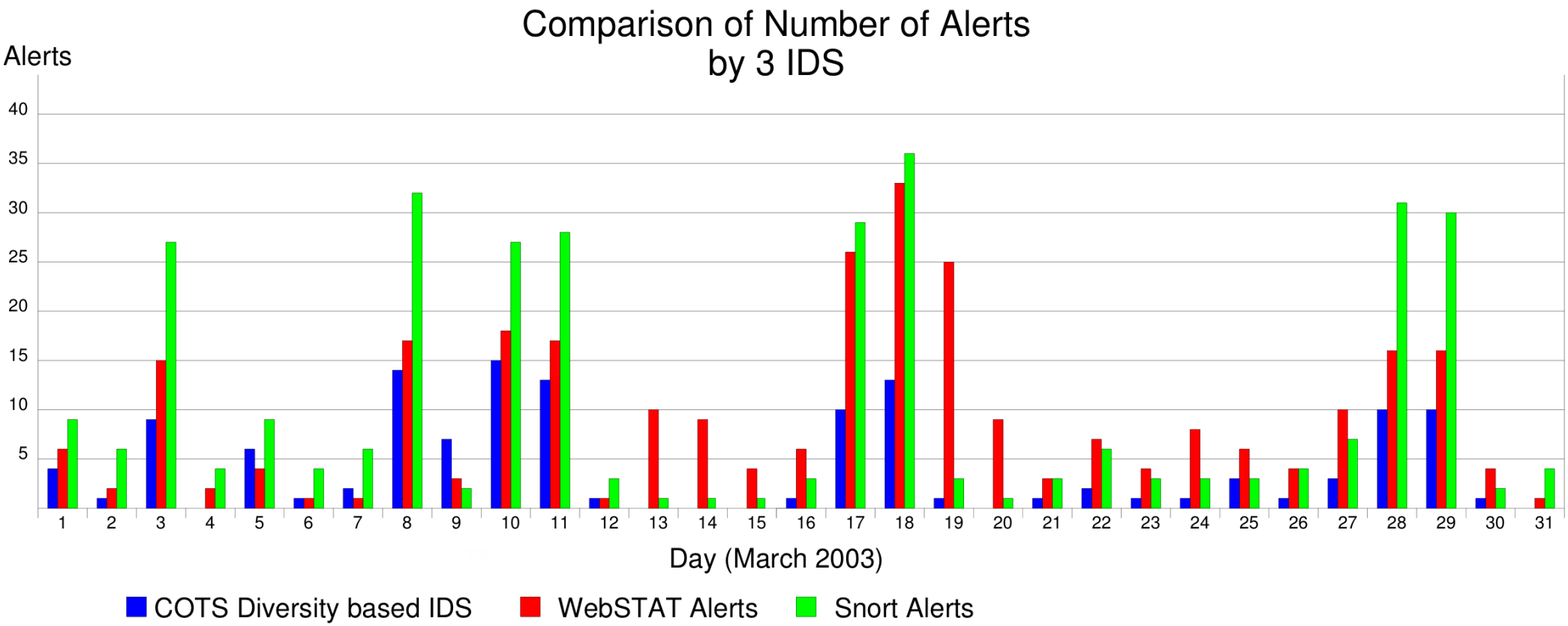
Masquage de différences de comportement:

- **36 règles pour éliminer des différences légitimes**

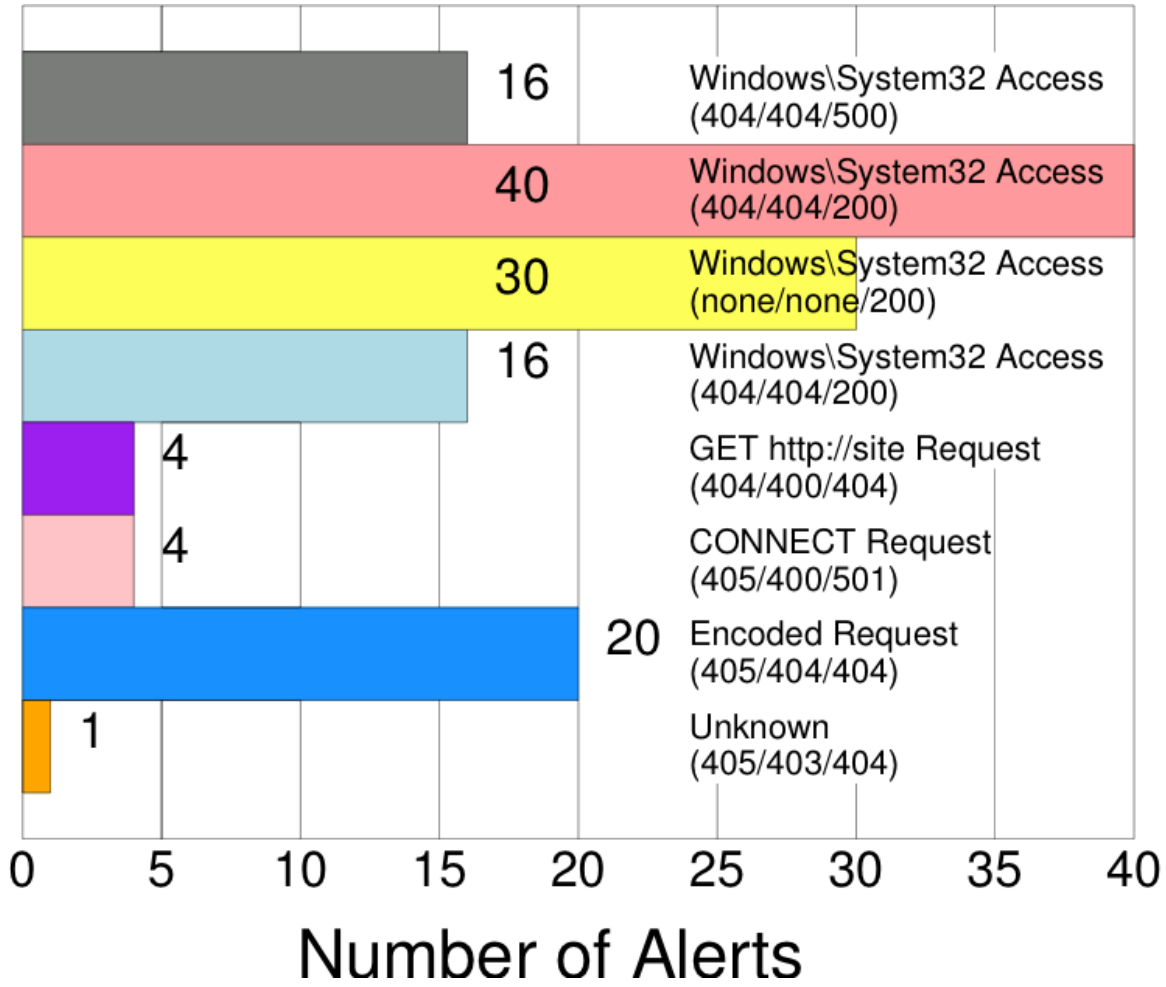
Analyse des différences en sortie



Comparaison avec d'autres IDS



Alertes émises



- **800 000 Requêtes**
- **131 Alertes**
- **102 Vrais Positifs**
- **29 Faux Positifs**

Conclusion sur l'approche « boite noire »

Toutes les sorties du composant ne sont pas comparées

- **Détection des erreurs qui se propagent sur la sortie HTTP**
- **Possibilité de manquer des attaques contre l'intégrité du serveur**
- **Trop peu d'informations pour correctement identifier le serveur compromis**

Donc …

Nécessité de mettre en place une approche « boite grise »

- **Ajout d'informations permettant**
  - **L'identification du serveur compromis**
  - **La détermination du type de l'alerte levée**
- **Augmenter la qualité de la détection (attaques contre l'intégrité)**

Approche boite grise: quelles informations comparer ?

IDEE 1: Comparaison de la suite des appels systèmes sur les différents serveurs

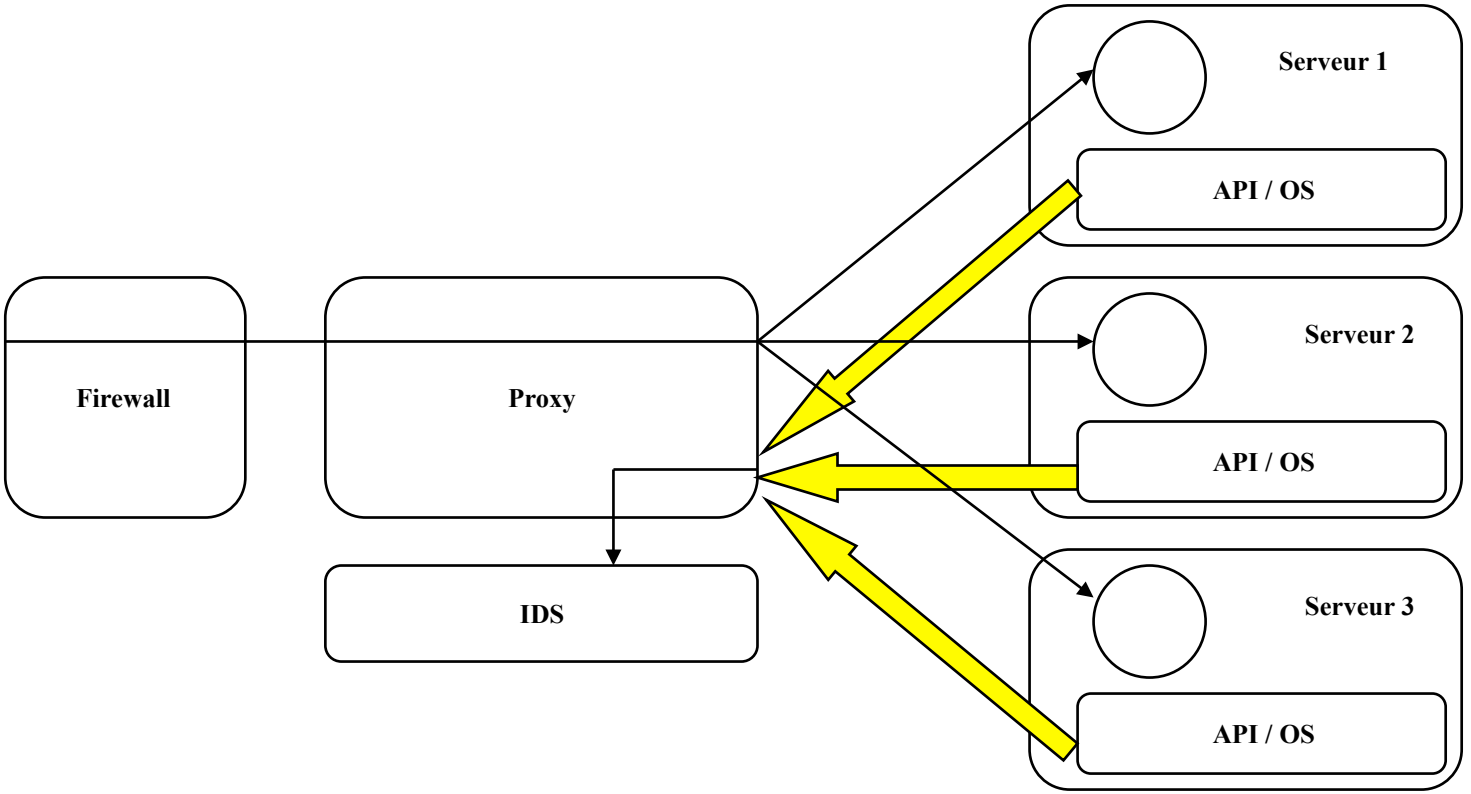
- **Problème: les APIs diffèrent sur les différents systèmes**
  - **Nommément**
  - **Fonctionnellement**
- **… Et n'ont pas forcément d'équivalent sur un autre système**
- **Gao et al. (RAID 2005 et 2006): notion de distance entre deux chaînes d'appels système équivalentes (i.e., qui réalisent la même fonction sur deux systèmes différents)**

Constat: les logiciels réalisent la même fonction, et génèrent des flux d'information proches

IDEE 2: une intrusion peut être caractérisée par un flux d'information illégal

- **Si les flux d'information sont sensés être identiques sur les différents serveurs, il suffit donc de les comparer pour détecter une intrusion**

Approche en « boite grise »



Génération des graphes de flux

Relation en terme d'information entre les détenteurs et les utilisateurs de l'information:

- **Les processus**
- **Les fichiers**
- **Les sockets**
- **…**

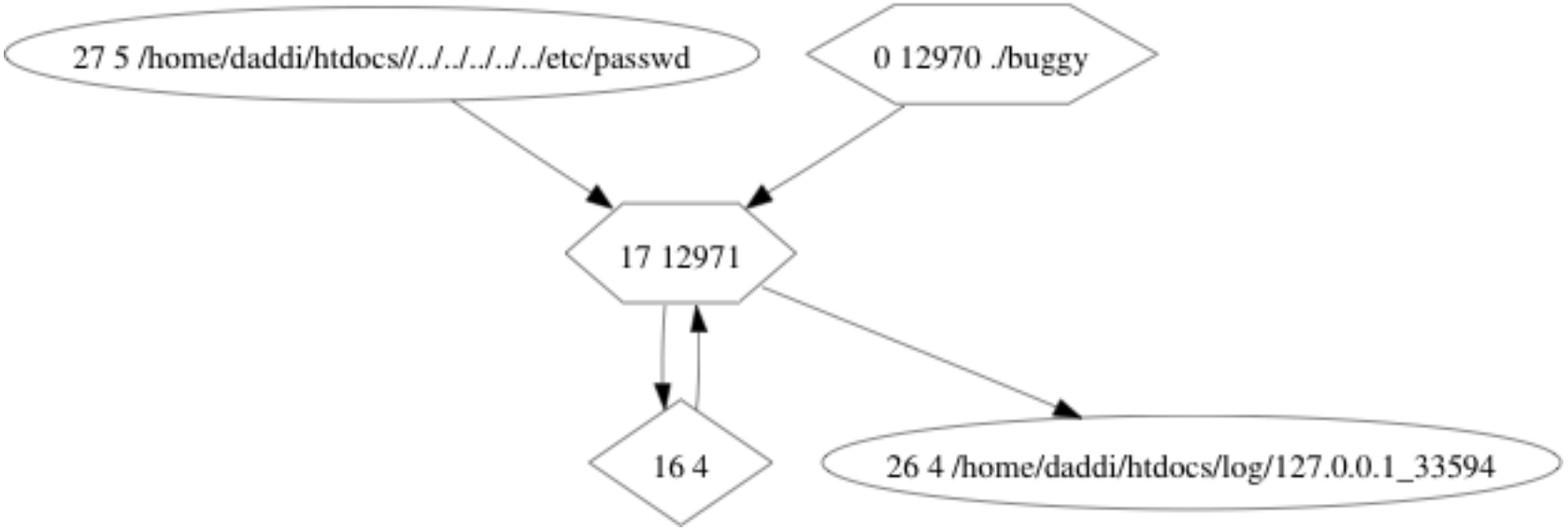
Graphes obtenus à partir des appels système

- **Identification des appels système générant des flux d'information**
- **Conséquence: seuls les appels système générant des flux d'information sont considérés**

Construction et comparaison des graphes

- **Mesure de similarité entre des graphes**

Exemple de graphe de flux



Graphe de flux et détection d'intrusion

Symptômes de différentes attaques

- **Attaque réussie contre la confidentialité: ajout de flux en lecture et/ou ajout d'objets**
- **Attaque réussie contre l'intégrité: ajout de flux en écriture et/ou ajout d'objets**

Détection par comparaison de graphes de flux:

- **Hypothèse de décorrélation des vulnérabilités**
- **La disparition ou l'ajout de flux ou d'objets n'affectera que le graphe de flux du serveur compromis**
- **Plus la similarité est importante, plus la probabilité qu'une intrusion ait réussi est faible**

Mesure de similarité entre graphes de flux (1)

Algorithme proposé par S. Sorlin, P.-A. Champlin, C. Solnon (Université de Lyon 1)

Soient deux graphes étiquetés G_1 et G_2

- $G_1=V_1, R_{V,1}, R_{E,1}$ et $G_2=V_2, R_{V,2}, R_{E,2}$
- V_i: ensemble des sommets
- $R_{V,i}$: ensemble des couples nœud/étiquette (v_i,l) avec $v_i \in V_i$ et l une étiquette
- $R_{E,i}$: ensemble des 3-uplets arrête/étiquette (v_i, v_j, l) avec $(v_i, v_j) \in V_i^2$ et l une étiquette
- $desc(G_i) = R_{V,i} \cup R_{E,i}$

Mapping

- Un mapping m entre G_1 et G_2 est un ensemble de couples de nœuds de G_1 et G_2: $m \subseteq V_1 \times V_2$

Similarité en fonction d'un mapping

- $Sim_m(G_1, G_2) = f(desc(G_1) \cap_m desc(G_2))/f(desc(G_1) \cup desc(G_2))$
- f fonction monotone croissante en fonction de l'inclusion
- f étroitement lié aux graphes considérés

Mesure de similarité entre graphes de flux (2)

Notion de splits

- La notion de mapping n'interdit pas qu'un nœud du graphe G_1 soit associé à plusieurs nœuds du graphe G_2
- *splits(m)*: l'ensemble des nœuds de G_1 et de G_2 qui ont plusieurs associations dans le mapping *m*
- $Sim_m(G_1, G_2) = [f(desc(G_1) \cap_m desc(G_2))-g(splits(m))]/f(desc(G_1) \cup desc(G_2))$
- g une fonction monotone croissante en fonction de l'inclusion

Mesure de similarité

- Calculer la similarité entre deux graphes, c'est calculer le maximum de la similarité pour tous les mappings possibles
- $Sim(G_1, G_2) = max_{m \subseteq V1 \times V2}[f(desc(G_1) \cap_m desc(G_2))-g(splits(m))]/f(desc(G_1) \cup desc(G_2))$

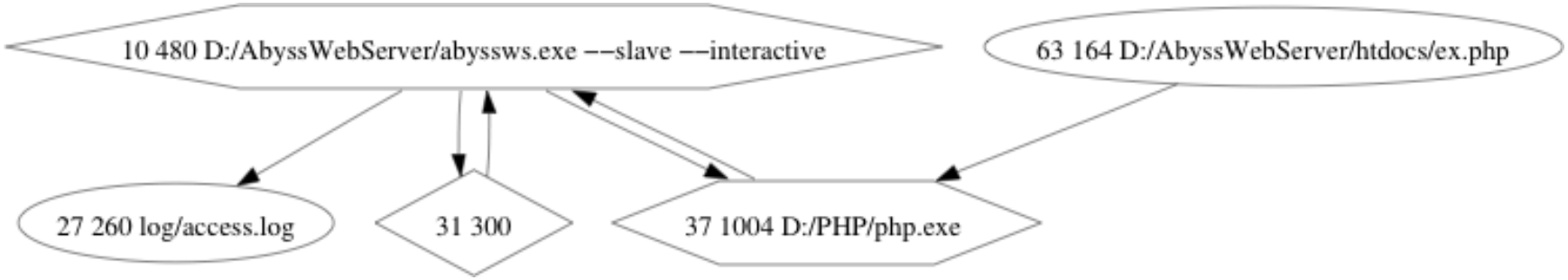
Optimisations pour les serveurs Web

Optimisations spécifiques au cas des serveurs web

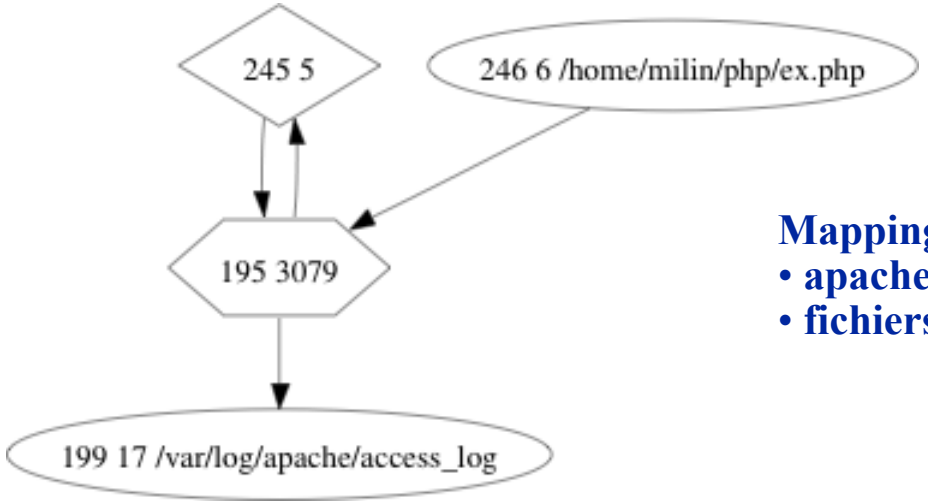
- Recherche de points communs entre les graphes
  - Socket recevant la requête
  - Processus lisant cette requête
  - Fichiers de log
  - Fichiers du répertoire racine du serveur web
- Interdiction de certains mappings
  - Grâce au typage des nœuds

Exemple concret: exécution d'un script PHP

Serveur Abyss, Windows 2000



Serveur Apache, Linux



Mapping:
- **apache et php.exe+abyssws.exe**
- **fichiers de log**

Détection d'intrusion et seuil de similarité

Relation détection d'intrusion/mesure de similarité

- Flux différents ou objets additionnels sont symptomatiques d'une intrusion
- La probabilité qu'une intrusion ait eu lieu est liée à la similarité des graphes
- Dans notre exemple, la similarité est comprise entre 0 et 1
  - 0: plus forte probabilité d'intrusion
  - 1: plus faible probabilité d'intrusion
- Comment déterminer un seuil à partir duquel on peut prétendre qu'il y a faible probabilité d'intrusion
  - Par l'expérience: 3 seuils définis pour les serveurs pris 2 à 2:
    - [0, 0.5] : une attaque a lieu
    - [0.5, 0.7] : indéterminé
    - [0.7, 1] : aucune attaque

Génération des alertes

s(1,2) / s(1,3)	[0, 0.5]	[0.5, 0.7]	[0.7, 1]
[0, 0.5]	A – A	A – Ind	A -NA
[0.5, 0.7]	Ind – A	Ind - Ind	Ind - NA
[0.7, 1]	NA – A	NA – Ind	NA - NA

Corrélation des méthodes de détection

Gray Box / Black Box	Alerte	Indéterminé	Pas d'alerte
Alerte	**Alerte**	**Alerte**	**Alerte**
Pas d'alerte	**Alerte**	**Alerte**	**Pas d'Alerte**

Supélec

Résultats

Environnement de test:

- **thttpd/Linux**
- **Apache/Mac-OS X**
- **Abyss/Windows**

Nouveau jeu de tests: 1 semaine de traffic HTTP (75000 requêtes)

	Black Box (avec masquage)	Gray Box (sans masquage)
Alertes	1	280 (0,4%)

Graphe et diagnostic

Diagnostic d'anomalies

- Identification du serveur compromis
- Expliquer la différence de sortie entre les COTS ou une similarité « faible » entre les graphes
- Objets non mappés dans le calcul de la similarité
  - Mise en évidence des flux ou objets additionnels, et donc des attaques contre la confidentialité ou l'intégrité

Conclusion

Méthode de détection d'intrusion

- Prometteuse, même en ne considérant que l'approche « boite blanche »
  - Peu de faux positifs
  - Peu ou pas de faux négatifs rencontrés
- Complète, si on utilise conjointement les deux approches proposées
  - Résultats en terme de faux positifs et négatifs à approfondir (masquage de différences légitimes à mettre en place)

Plan de la présentation: Détection par invariants

- **Introduction à la détection d'intrusion au niveau applicatif**
  - ❑ **Etat de l'art**
- **Modèle construit par analyse statique**
- **Modèle construit par analyse dynamique**

Détection comportementale au niveau applicatif

- Via les interactions application/système
  - Observation du processus par le système
  - Séquences d'appels système

- A l'intérieur de l'application
  - Détection d'erreurs dans l'application
  - Les erreurs se caractérisent de la même manière que pour les fautes accidentelles

Erreurs détectables dans l'application

- **Erreur dans le flot d'exécution**
  - **Les instructions, appels de fonctions ou appels système ne s'exécutent pas dans le bon ordre**

- **Erreur au niveau des données**
  - **Le flot de données est incorrect (des variables sont modifiées par d'autres variables de façon inattendue)**
  - **Des variables prennent des valeurs incorrectes**

Méthodes de détection

- **Violation de l'intégrité du flot d'exécution**
  - **La majorité des travaux existants**
  - **Sauf quelques rares exceptions, ne détecte pas les attaques contre les données**

- **Violation de l'intégrité du flot de données**
  - **Très efficace pour détecter les attaques usuelles contre les données (buffer overflow, etc.) … Mais très lent à l'exécution**

Nos travaux

- **Modèle de détection comportementale autour des données**

- **Objectif**
  - **Détecter des valeurs incorrectes de données**
  - **Faible surcoût à l'exécution**

- **Moyens**
  - **Découverte d'invariants, analyse statique, analyse dynamique**

Plan: Analyse statique

- **Analyse statique**
  - **Introduction à la notion d'invariant**
  - **Définition du modèle de comportement normal de l'application**
  - **Implémentation et évaluation**

Supélec

Introduction à la notion d'invariant (1)

```
1 - int auth_ok = 0;
2 - if(passwd != NULL)
3 – while(auth_ok != 1){
4 —— type = packet_read(data);
5 —— switch (type) {
6 —— ...
7 —— case SSH_CMSG_AUTH:
8
9 ——— auth_ok = auth(passwd, data);
10 ——— break;}
11
12 - authenticated(uid);
```

Introduction à la notion d'invariant (2)

```
1 - int auth_ok = 0;
2 - if(passwd != NULL)
3 — while(auth_ok != 1){
4 —— type = packet_read(data);
5 —— switch (type) {
6 —— ...
7 —— case SSH_CMSG_AUTH:
8 ——— assert(passwd != NULL)
9 ——— auth_ok = auth(passwd, data);
10 ——— break;}
11 - assert((auth_ok == 1 && type == SSH_CMSG_AUTH) || auth_ok == 0)
12 - authenticated(uid);
```

Supélec

Introduction au modèle

- **Objectif de l'attaquant:**
  - **Donner des valeurs incorrectes à des données**

- **Objectif de la détection:**
  - **Détecter des appels système illégaux d'un point de vue des données du programme**

Supélec

Modèle de comportement

- **Modèle de comportement:**
  - **MC = $U_i(Sc_i, V_i, C_i)$**

- **Pour un appel système Sc_i**
  - **V_i = Ensemble des variables qui influencent l'appel**
  - **C_i = ensemble des contraintes calculées sur Vi (invariants, ici contraintes sur les valeurs des données)**

Framework d'analyse statique

Outil d'analyse de programmes en C

- **Frama-C (développé par le CEA)**
- **Développement d'un plugin pour Frama-C: coninva**

Utilisation de plugins existants

- **PDG (Program Dependency Graph)**
- **Value Analysis Plugin (calcul de valeurs de variables)**

Supélec

Calcul de V_i

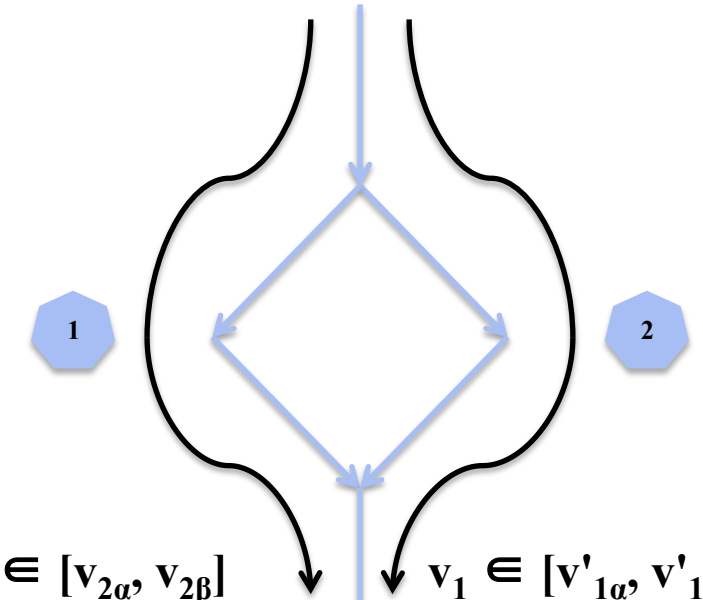
- Calcul de l'ensemble des variables qui influencent un appel système SC_i
  - Utilisation du PDG de Frama-C (cf. technique de slicing)
  - Récupération de toutes les variables dans le PDG avec pour point d'intérêt un appel système SC_i

Supélec

Calcul d'invariants (1)

- **Utilisation du Value Analysis Plugin**
  - **Obtention d'invariants de la forme**
    - $\bigwedge_i v_i \in [v_{i\alpha}, v_{i\beta}]$
  - **Perte d'information de la relation entre valeurs de variables sur des chemins différents (ex: $(a==1 \wedge b==2) \vee (a==0 \wedge b==1)$)**
  - **Inconvénient dû à l'algorithme du Value Analysis Plugin qui par défaut unifie toutes les valeurs pour chaque variable**

Calcul d'invariants (2)



$v_1 \in [v_{1\alpha}, v_{1\beta}] \wedge v_2 \in [v_{2\alpha}, v_{2\beta}]$ $\quad$ $v_1 \in [v'_{1\alpha}, v'_{1\beta}] \wedge v_2 \in [v'_{2\alpha}, v'_{2\beta}]$

$(v_1 \in [v_{1\alpha}, v_{1\beta}] \wedge v_2 \in [v_{2\alpha}, v_{2\beta}]) \vee (v_1 \in [v'_{1\alpha}, v'_{1\beta}] \wedge v_2 \in [v'_{2\alpha}, v'_{2\beta}])$

F(…)

Calcul d'invariants (3)

- **Plugin coninva**
  - **Utilise l'option slevel du Value Analysis Plugin**
  - **Utilise les mécanismes d'interception de Frama-C**
  - **Récupère les informations internes du Value Analysis Plugin avant unification**

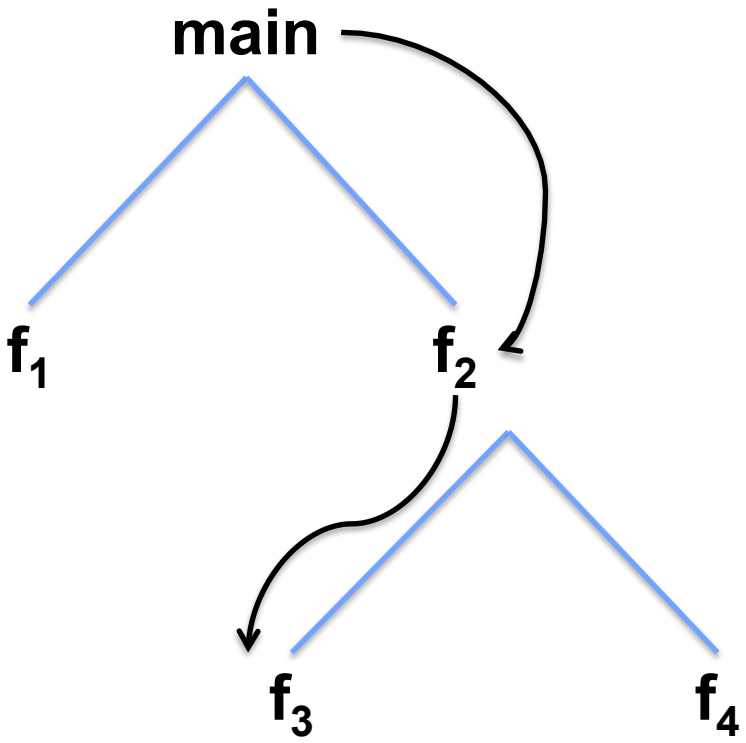
- **Invariants obtenus de la forme**
  - **(a == 1 ∧ b==2) ∨ (a==0 ∧ b==1)**

Supélec

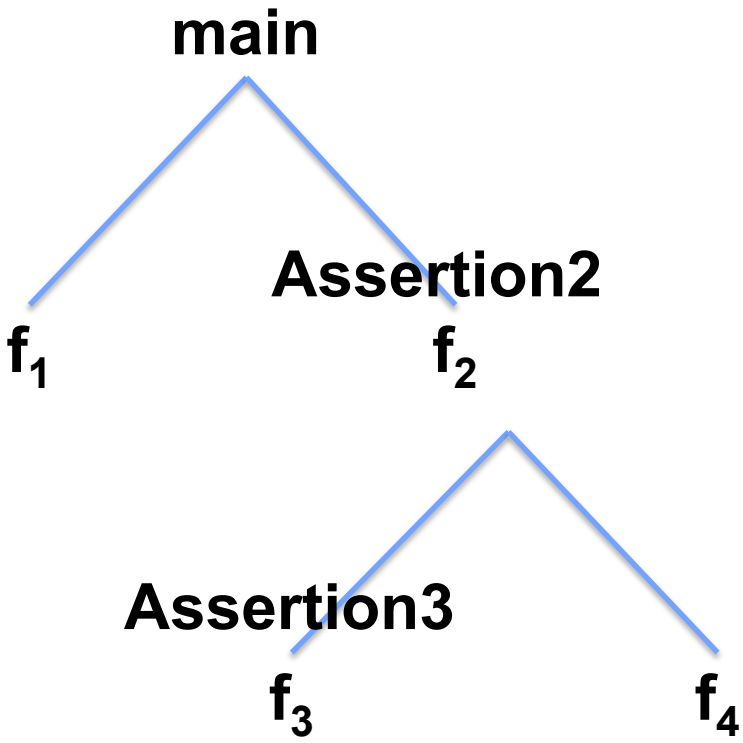
Assertions exécutables

- Chaque invariant calculé est tissé dans le code sous forme d'assertion exécutable
- Difficile de savoir quelle fonction contient un appel système + seules les variables locales sont visibles à un point donné
  - Calcul d'invariants pour chaque appel de fonction
  - Insertion d'une assertion exécutable devant chaque appel de fonction

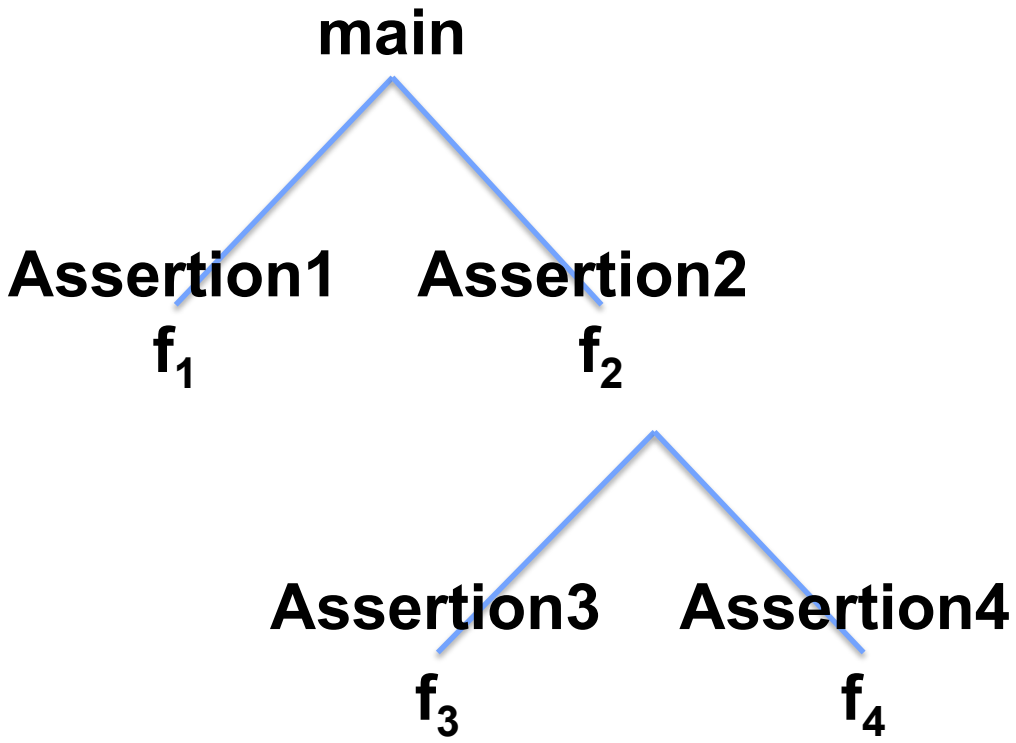
Ajout d'assertions exécutables



Ajout d'assertions exécutables



Ajout d'assertions exécutables



Supélec

Résultats de l'instrumentation

	Dropbear	ssmtp	fnord	ihttpd	nullhttpd
Lignes de code	11177	2717	2622	1180	5968
Nombre d'appels de fonctions	429	314	232	289	399
Durée d'analyse (mn)	16h	19mn	3h45mn	1mn	5h17mn
Nombres d'assertions	257	237	6	109	234
Nombre de points d'injections	371	261	209	232	356

Supélec

Performances (SPEC CPU 2006)

	Lbm	Mcf	Libquantum	Bzip2	milc	sjeng
Score non instrumenté	19.17	17.72	23.45	16.54	12.59	18.24
Score instrumenté	19.24	17.80	23.62	16.59	12.64	18.41
Perte de score	0.37%	0.45%	0.72%	0.30%	0.40%	0.93%

Evaluation

- **Evaluation de la couverture de la détection**
  - **Par injection de faute**
- **Simulation du résultat d'une attaque**
  - **Modèle d'attaque: modification équiprobable de n'importe quelle variable influençant un appel de fonction**
  - **Valeur aléatoire imposée à la variable**

Supélec

Taux de détection obtenus

	ihttpd	nullhttpd
Nombre d'injections réalisées	56584	96925
Nombre d'injections détectées	6419 11.3%	29107 30.3%
Nombre d'alertes levées	27938	1072310
Nombre moyen d'alertes levées par injection détectée	4.35	36.84

Comparaison avec un IDS plus classique

		coninva	Syscall anomaly
ihttpd	Injections détectées	6419	4386
	Taux de détection	11,34%	7,75%
nullhttpd	Injections détectées	29107	27080
	Taux de détection	30,03%	27,94%

Conclusion première partie

Faible taux de détection …

- **Toutefois supérieur aux autres méthodes « classiques » qui n'adressent pas spécifiquement les attaques contre les données**
- **Pourrait être amélioré en calculant d'autres types d'invariants**

… mais très faible dégradation des performances !

Conclusion générale
Bilan

- **Deux grands types de mécanismes de détection d'erreur adaptés avec succès à la détection d'intrusion**
  - **Diversification fonctionnelle**
  - **Contrôles de vraisemblance**
- **Implémentation et évaluation de chacune des approches en présence d'attaques**
- **Approches complémentaires (sources de données diversifiées)**

Quelques Publications relatives à ces travaux

[RAID 2005] **Eric Totel, Frédéric Majorczyk, and Ludovic Mé.** ***COTS diversity based intrusion detection and application to web servers*****.**

[IFIP SEC 2008] **Frédéric Majorczyk, Eric Totel, Ludovic Mé, and Ayda Saidane.** ***Anomaly detection with diagnosis in diversified systems using information flow graphs*****.**

[IFIP SEC 2011] **Jonathan-Christofer Demay, Frédéric Majorczyk, Eric Totel, and Frédéric Tronel.** ***Detecting illegal system calls using a data-oriented detection model*****.**

[CRISIS 2012] **Romaric Ludinard, Eric Totel, Frédéric Tronel, Vincent Nicomette, Mohamed Kaâniche, Eric Alata, Rim Akrout, Yann Bachy.** ***Detecting Attacks against Data in Web Applications.***