Octopus password breaker

14 avril 2015

Danny Francis
<danny.francis@telecom-paristech.fr>

Introduction

- Un espace de recherche extrêmement grand, dont la taille augmente exponentiellement par rapport aux tailles des mots de passe
- Impossible de le parcourir exhaustivement
- Méthodes pour optimiser la recherche :
  - *Tester d'abord les mots de passe les plus probables (dictionnaires, masques) => Outils usuels tels que John the Ripper, Hashcat ou oclHashcat*
  - *Augmenter la puissance de l'ordinateur => le coût d'un ordinateur augmente bien plus vite que sa puissance*
  - *Combiner plusieurs machines => moins coûteux, mais nécessite un intermédiaire pour les faire communiquer entre elles*
- La première et la troisième méthode sont utilisées dans Octopus

État de l'Art

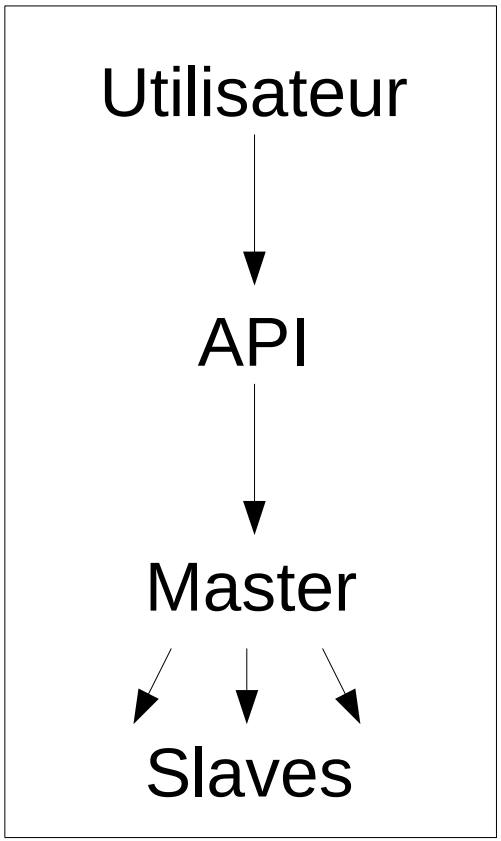
- John the Ripper : casseur de mots de passe *open source*
- Distribuer des attaques avec John the Ripper :
  - *Utilise la norme MPI*
  - *Pas de tolérance de fautes*
- Hashcat/oclHashcat : gratuit mais pas *open source*
- Distribuer des attaques avec Hashcat ou oclHashcat :
  - *Hashtopus*
  - *Ne fonctionne qu'avec Hashcat*

État de l'Art

- Elcomsoft Distributed Password Recovery
  - *Apparemment très efficace*
  - *Pas open source = impossible d'ajouter de nouveaux algorithmes*
  - *$$$: 599 euros pour 5 machines, 1999 euros pour 20 machines, 4999 euros pour 100 machines*
- Distributed Hash Cracker par Andrew Zonenberg
  - *Papier universitaire à propos d'un casseur de mots de passe distribué*
  - *CPU + GPU*
  - *Tolérance de fautes*
  - *Seulement MD5*
- MWR Distributed Hash Cracking on the Web
  - *WebCL => n'importe quelle machine pourra l'utiliser*
  - *Mauvaises performances selon les mesures de MWR Labs à cause de la lenteur de Javascript : 15 fois plus lent que oclHashcat*

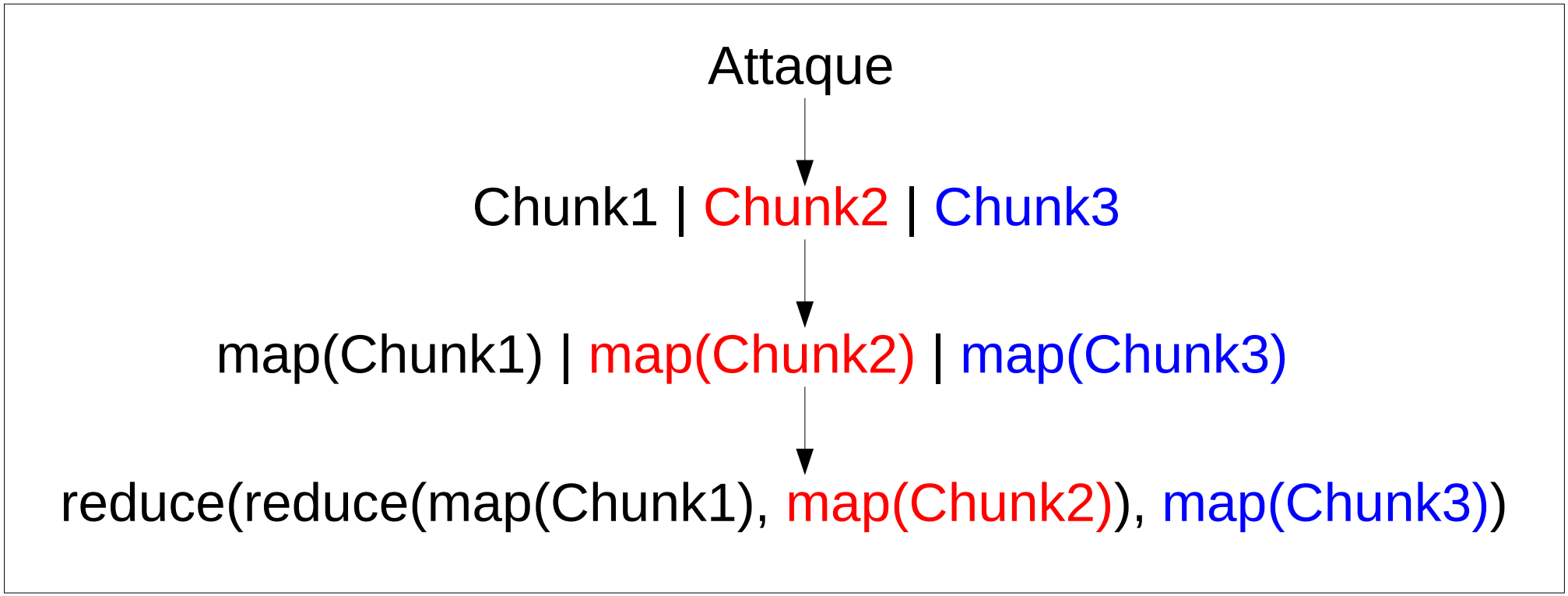
Présentation d'Octopus

- Octopus : faire travailler ensemble des machines qui savent travailler séparément
- Architecture « *master-slave* »



Présentation d'Octopus

- Distribuer et récupérer des résultats : MapReduce
  - *Division de l'attaque en attaques moins coûteuses*
  - *Les nœuds dits esclaves réalisent les petites attaques (fonction « map »)*
  - *Les résultats sont combinés en une seule liste (fonction « reduce »)*



Présentation d'Octopus

- Les nœuds communiquent entre eux avec ce type de messages :

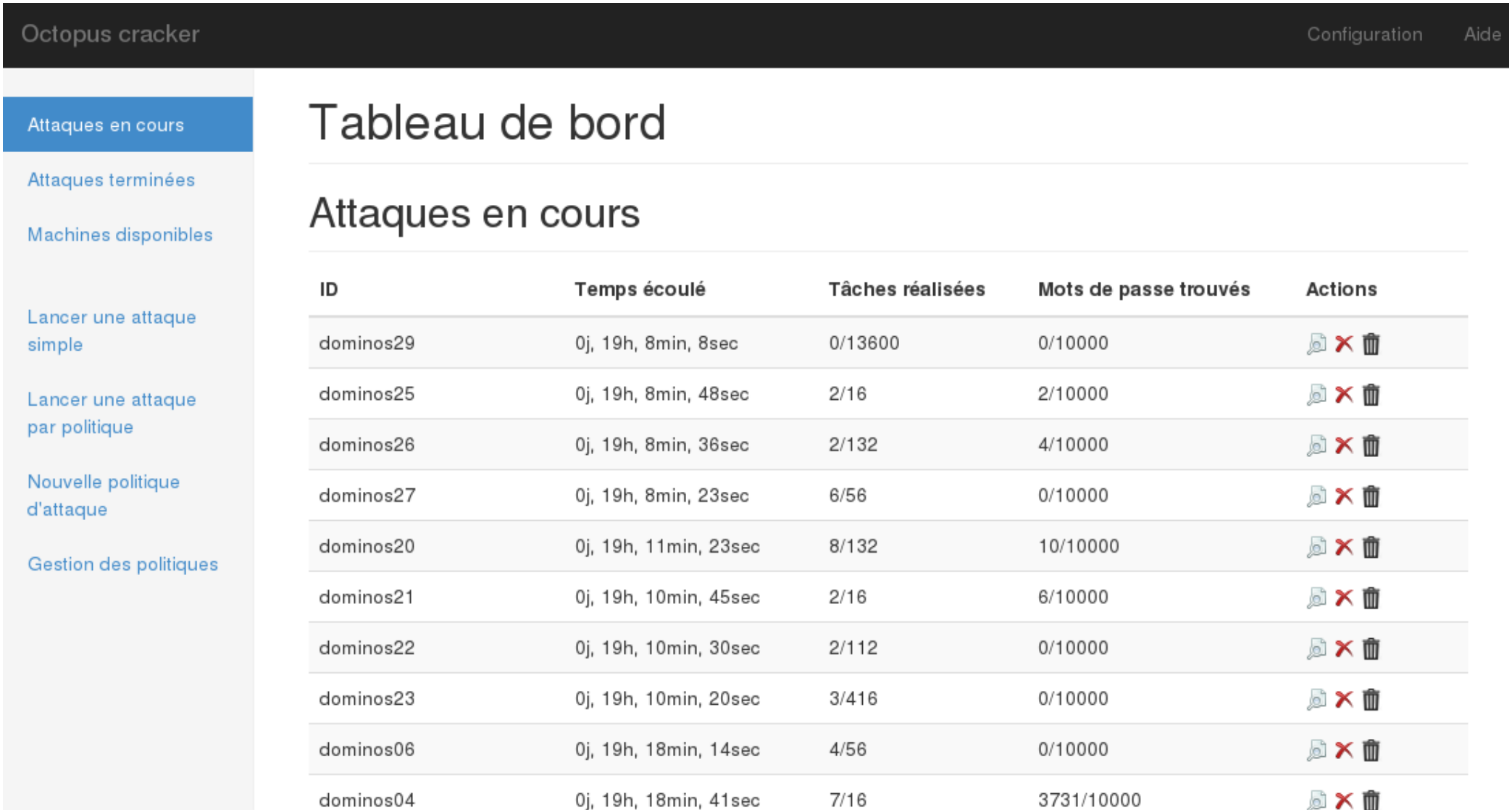
FONCTION arg1 arg2

- Messages empaquetés dans des JSON

[FONCTION, arg1, arg2]

- Envoyés par requêtes HTTP GET
- UI : Interface Web

Présentation d'Octopus



Présentation d'Octopus

- Fonctions d'Octopus :
  - *NODETYPE : identifie le type de nœud*
  - *PUT : envoie un objet (généralement un hash ou une petite attaque)*
  - *GET : demande une information*
  - *DO : envoie une attaque*
  - *RESULT : envoie un résultat (pour une attaque mais aussi pour une requête GET)*
  - *REMOVE : supprime un objet*
  - *OK : action réalisée avec succès*
  - *ERROR : erreur*

Implémentation

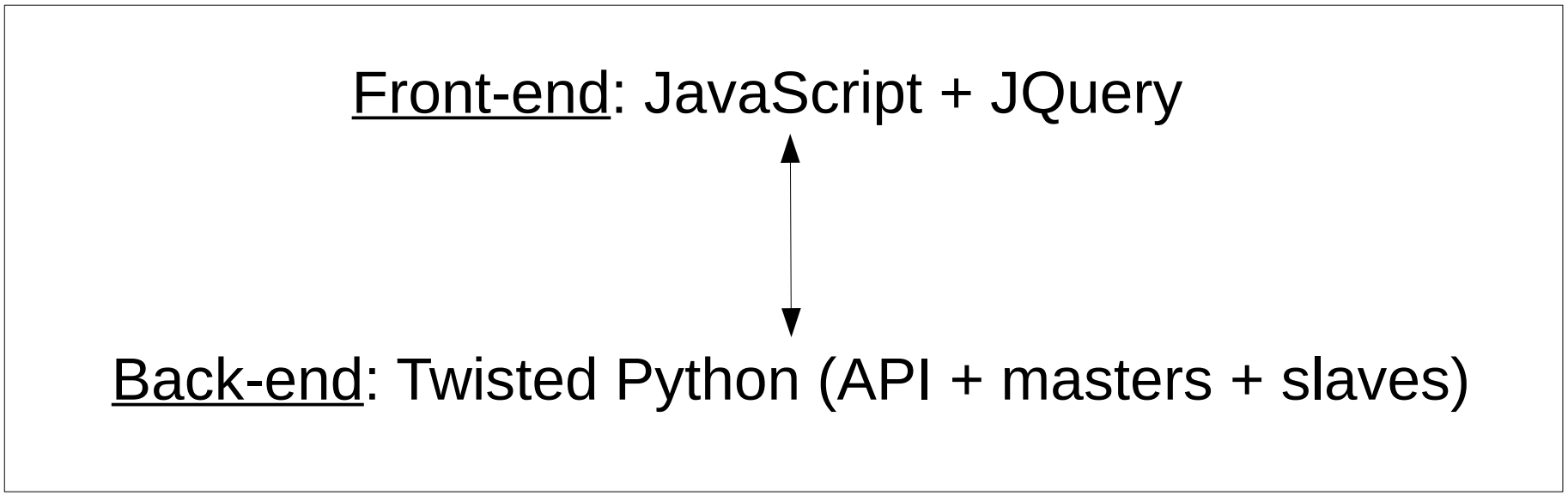
- Partie *back-end* : Python
- Twisted Python
  - *Programmation asynchrone*
  - *Deferreds, callbacks*

Implémentation

- Deux types d'objets pour chaque nœud : les Factory et les Protocol
- Factory:
  - *Caractéristiques liées au nœud enregistrées dans ses attributs*
  - *Crée un objet Protocol quand un autre nœud s'y connecte*
- Protocol:
  - *Empaquette les requêtes dans des JSON et des requêtes HTTP GET*
  - *Lit les requêtes reçues et déclenche les actions correspondantes*

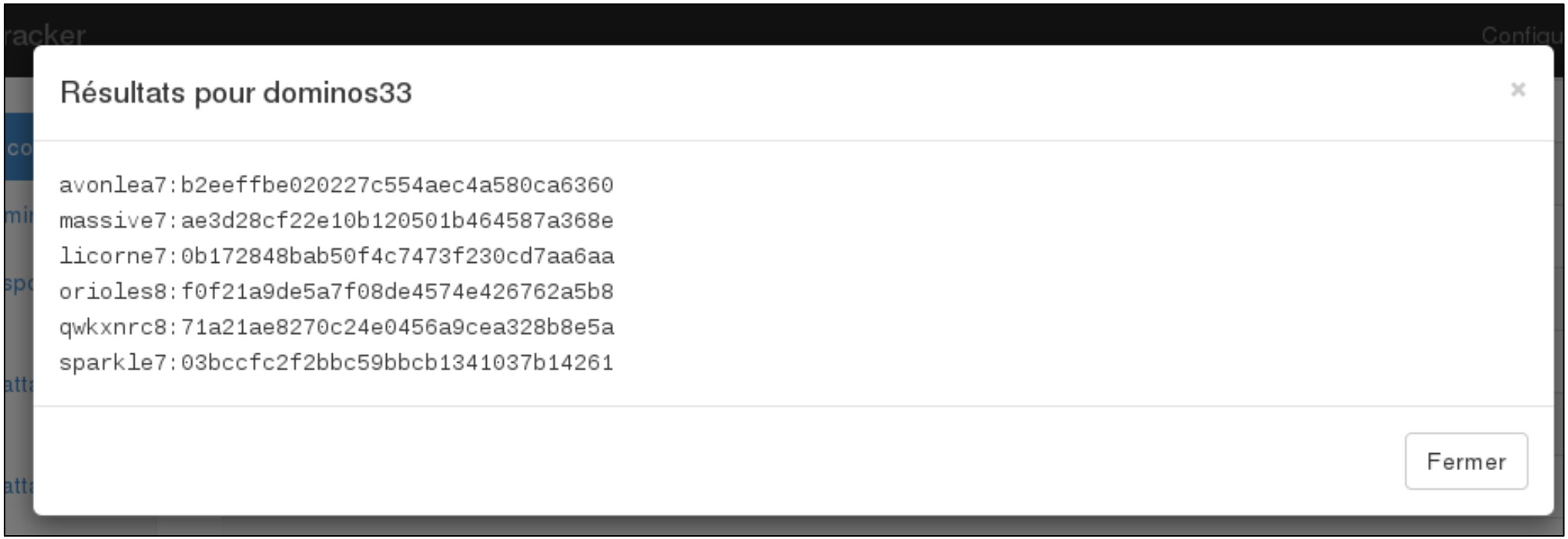
Implémentation

- Interface réalisée avec un template Bootstrap
  - Bootstrap : Ensemble d'outils Web réalisé par Twitter
- Interactions avec l'utilisateur : JavaScript + JQuery



Implémentation

- Informations disponibles



Implémentation

- Informations disponibles

Attaques terminées

ID	Temps écoulé	Mots de passe trouvés	Actions
dominos24	0j, 2h, 58min, 57sec	22/10000	
dominos32	0j, 2h, 29min, 10sec	24/10000	
dominos07	0j, 19h, 3min, 8sec	0/10000	

Attaques arrêtées

ID	Temps écoulé	Mots de passe trouvés	Actions
Aucune tâche arrêtée			

Implémentation

- Informations disponibles

Implémentation

- Politiques d'attaque

Nouvelle politique d'attaque

Nom de la politique

new_policy | Créer la politique

Attaques

mask | ?a?a?a?a?a?a?a?a | Ajouter l'attaque

Ordre	Type d'attaque	Attaque	Supprimer
1	toggle_case	john	✖
2	dictionary	rockyou	✖
3	mask	?a	✖
4	mask	?a?a	✖
5	mask	?a?a?a	✖

Implémentation

- Politiques d'attaque

Implémentation

- Attaques possibles
  - *Attaques simples*
    - Dictionnaires
    - Masques
    - Variantes (toggle-case, etc.)
  - *Politiques d'attaques*
    - Séquence d'attaques simples
    - Ex : attaque par dictionnaire, puis toutes les chaînes de minuscules de longueur 8, etc.
  - *Attaque automatique (Markov)*

Implémentation

- Chaînes de Markov
- Modèle probabilistique
  - *Étant donnés un système et un ensemble d'états, la probabilité de passer à un état à partir d'un autre état ne dépend pas des états parcourus auparavant*
  - *Ex : un nageur nage contre le courant, et vérifie sa progression toutes les 30 secondes*
    - *État a : il a avancé*
    - *État b : il n'a pas bougé*
    - *État c : il a reculé*
    - *Au temps t : il est à l'état a*
    - *Chaînes de Markov : la probabilité de passer à l'état b trente secondes plus tard ne dépend pas de comment le nageur a nagé auparavant*

Implémentation

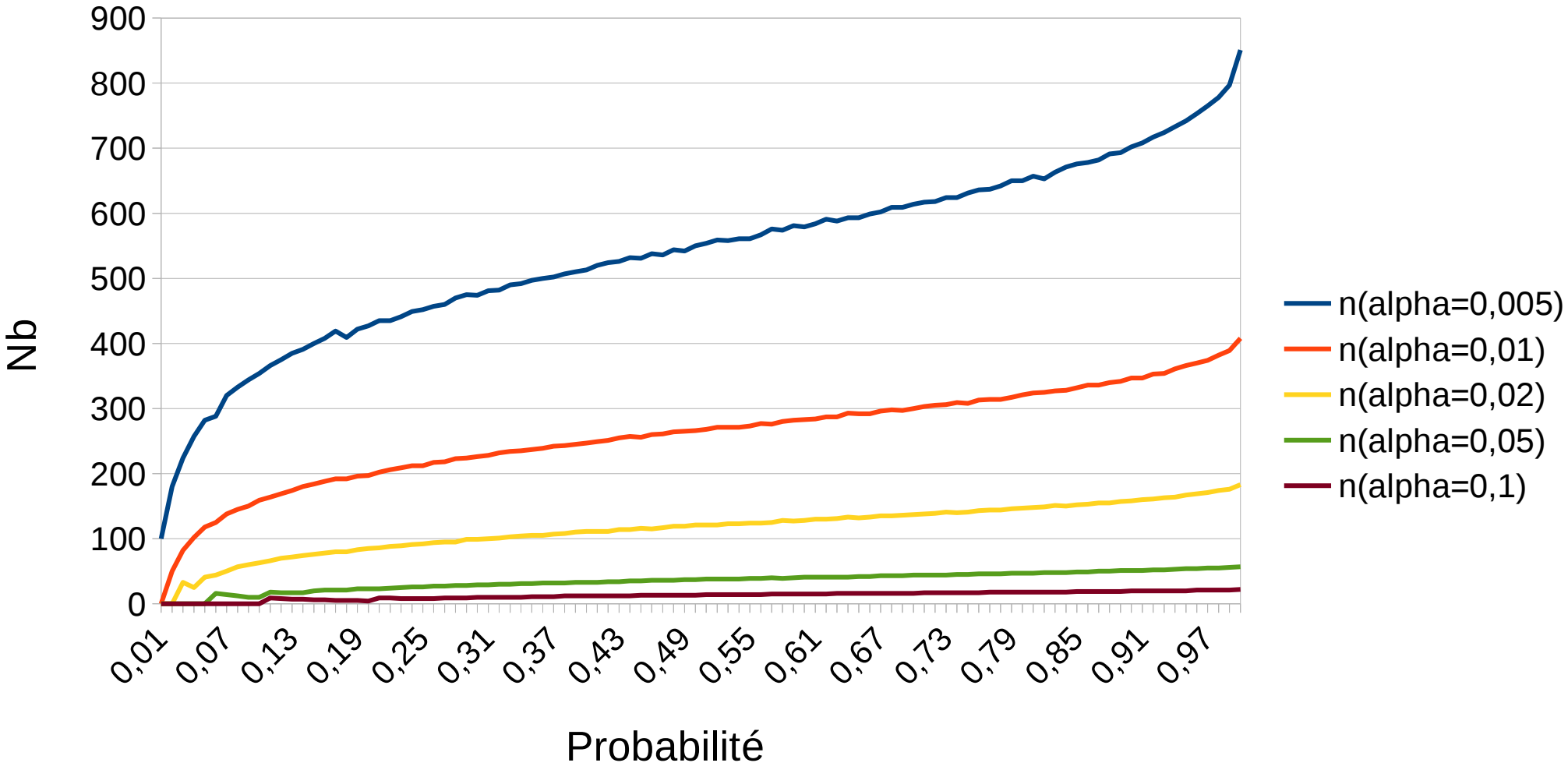
- Mode Markov de John the Ripper
  - *Probabilité d'obtenir un h après un c supérieure à la probabilité d'obtenir un h après un m*
- Mode automatique d'Octopus
  - *Probabilité qu'une attaque donnée fasse passer le pourcentage de mots de passe découverts de i % à j %*
  - *Processus de décision markovien : (E, A, P)*
    - *E = entiers compris entre 0 et 100 (pourcentage de mots de passe découverts)*
    - *A = ensemble d'attaques simples*
    - *P = fonction qui associe P(a, i, j) à une attaque a, un entier i et un entier j, avec P(a, i, j) étant la probabilité que a fasse passer le pourcentage de mots de passe découverts de i % à j %*
  - *Notre but : déterminer P*
    - Grâce à *P* et au temps moyen mis par des attaques simples, on peut déterminer la politique d'attaque qui serait la meilleure compte tenu d'une deadline donnée

Implémentation

- Comment déterminer P
  - *Apprendre grâce aux attaques réalisées par Octopus sur plus de 1000 hashs*
    - Pour chaque attaque : tableau à deux dimensions *T* tel que T[i, j] = probabilité d'augmenter le pourcentage de mots de passe découverts de i % à j %
    - Si une attaque fait passer ce pourcentage de i % à j % :
      - $T[i, j] = T[i, j] \times (1 - \alpha) + \alpha$
      - Pour $j' \neq j$: $T[i, j'] = T[i, j'] \times (1 - \alpha)$
    - Avec α un paramètre à choisir entre 0 et 1
      - Si α est très petit, convergence lente mais bonne précision à long terme
      - Si $\alpha \approx 0.1$, convergence rapide mais mauvaise précision à long terme
    - Question : quelle valeur donner à α ?

Implémentation

- Convergence



Tests

- Passage à l'échelle ?
- Trois machines
  - Machine 1: oclHashcat, AMD Radeon HD 6630
  - Machine 2: Hashcat, quad-core Intel Core i5-3320M CPU @ 2.60GHz
  - Machine 3 : Hashcat, dual-core Intel Core 2 Duo CPU P8600 @ 2.40GHz
- Communiquent *via* le réseau Wifi HSC

Tests

- Casser 15000 hashs en testant toutes les chaînes de minuscules de longueur 9
- Test de chaque machine séparément avec son propre outil, puis ensemble avec Octopus
- Après une heure de cassage :
  - *Mesure du pourcentage de l'espace de recherche couvert par chaque machine séparément*
  - *Mesure du pourcentage de l'espace de recherche couvert par le cluster entier*
  - *Soient M1, M2, M3 les pourcentages des machines 1, 2 and 3, et MC le pourcentage pour le cluster. Avec un passage à l'échelle, nous devrions avoir :*

M1 + M2 + M3 = MC

Tests

- Résultats
  - *Machine 1 : 17.7 %/h*
  - *Machine 2 : 1.5 %/h*
  - *Machine 3 : 0.4 %/h*
  - *Cluster : 19.3 %/h (théorie : 19,6%/h)*
- Le passage à l'échelle semble être réalisé
- Quelles limites...
  - *Faible nombre de machines*
  - *Attaque par masque : peu d'échanges de données → une attaque par dictionnaire serait bien moins efficace si les chunks devaient être échangés*
  - *Question : en supposant le passage à l'échelle réalisé, quand est-ce qu'une attaque par dictionnaire est plus efficace avec un cluster qu'avec une simple machine ?*

Tests

- Pour la machine 1 (Mach1)
  - *483104 MD5 hashs, attaque par dictionnaire avec RockYou (size ~ 100 MB et des mots de longueur moyenne ~ 10) : 7 secondes*
- Quand est-ce qu'un cluster de Mach1 est plus efficace qu'une simple Mach1 ?
  - *Soit t_m le temps moyen pris pour tester un mot, soit n_c le nombre de chunks, soit C la taille d'un chunk, m la taille d'un mot et soit T_1 le temps mis par Mach1 :*

$$T_1 = (t_m \times n_c \times C)/m$$

  - *Soit D le débit moyen entre deux machines, soit n_n le nombre de nœuds et soit T_2 le temps mis par le cluster :*

$$T_2 = [(n_c \times C)/n_n] \times [t_m/m + 1/D]$$

Tests

- But : T1 < T2
- Application numérique
  - $t_m = 7 \times 10^{-7}$ *secondes*
  - $D = 5 \times 10^{6}$ *octets/seconde*
  - $m = 10$ *octets*
  - *Résultat :*

$$n_n \geq 4$$

  - *Mauvaises conditions : on peut supposer qu'un cluster de quatre machines sera toujours (beaucoup) plus efficace qu'une machine seule*

Limites

- Nombre maximum de machines ?
- Attaques par dictionnaire longues
- Pas de communications sécurisées → à utiliser dans un réseau isolé avec des utilisateurs de confiance...

À venir

- Aux alentours du 15 mai : documentation, et quelques améliorations
  - Nombre de divisions par attaque et par type de hash
  - Identifier les hashs et attaques disponibles pour chaque slavenode
  - Donner la possibilité de placer des dictionnaires directement sur les slavenodes
  - Types de hashs supplémentaires
- Fin juin ou début juillet : l'implémentation complète de l'attaque automatique

Conclusion

- Casseur de hashs flexible et distribué
- Une caractéristique originale : un mode automatique reposant sur un processus de décision markovien
- Quelques limites, en particulier concernant la sécurité des communications
- Open-source, sur GitHub : https://github.com/hsc-lab/octopus

Remarques, questions

???