Qui suis-je ?

- Consultant en sécurité applicative depuis 2009
  - Tests de pénétration applicatifs (ethical hacking);
  - Revues de code de sécurité;
  - Accompagnement d'équipe de développement dans la réalisation d'applications sécuritaires;
  - Élaboration de cadre normatifs de développement sécuritaire.
- Avant ça
  - Maîtrise en informatique portant sur la sécurité applicative (model-checking), Université Laval 2009;
  - Baccalauréat intégré en mathématiques-informatique, Université Laval, Université Laval 2006.
- Autres
  - Donne des présentations sur la sécurité applicative;
  - Participe à des compétitions de sécurité.

ReDoS

Hackerspace mai 2011

Quand la validation de données tourne mal

Motivations

- Les expressions régulières sont partout !
  - Navigateurs Web
  - FireWalls
  - Serveurs Web
  - IDS/IPS
  - Applications Web
  - Etc

Motivations

- Connaissance limitée des expressions régulières et des concepts théoriques sous-jacents
- Utilisation d'algorithmes de validation non efficaces
- Définition d'expressions régulières complexes

Plan

Retour sur les REGEX

CAUSES dES REDOS

Solutions POSSIBLES

CONCLUSION

Retour sur les regex

Expressions régulières

- Une expression régulière permet de valider l'occurrence de caractères et l'ordre dans lequel ces caractères apparaissent dans un mot.

- Une expression régulière est définie à partir de caractères et d'opérateurs.

Expressions régulières (suite)

e1 | e2	**Choix entre deux expressions régulières**
e1e2	**Séquence de deux expressions régulières**
e*	**Répétition d'une expression régulière zéro ou plusieurs fois**
e?	**Expression régulière présent zéro ou une fois**
.	**Désigne n'importe quel caractère**
[ ]	**Choix entre une série de caractères**

Expressions régulières étendues

- Au fil du temps des ajouts ont été faits aux expressions régulières

- Références ultérieures

(.*)\1

papa, bonbon

  - Il ne s'agit plus d'expressions régulières proprement dites

CauseS des ReDOS

Mise en situation

- Expression régulière « simple »

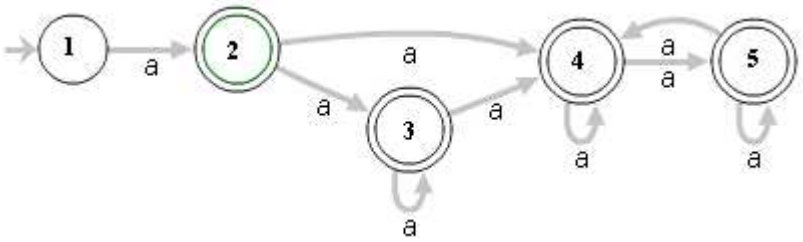
^(a+)+$

- Chaîne de caractères à valider

aaaaaaaaaaaaaaaaX

Algorithme de reconnaissance naïf

1. Traduction de l'expression régulière en automate fini non déterministe (NFA)

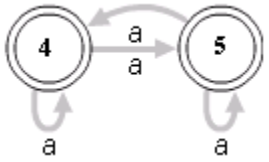


2. Parcours l'automate en conservant en mémoire tous les chemins possibles

65536 chemins possibles

Algorithme de reconnaissance naïf

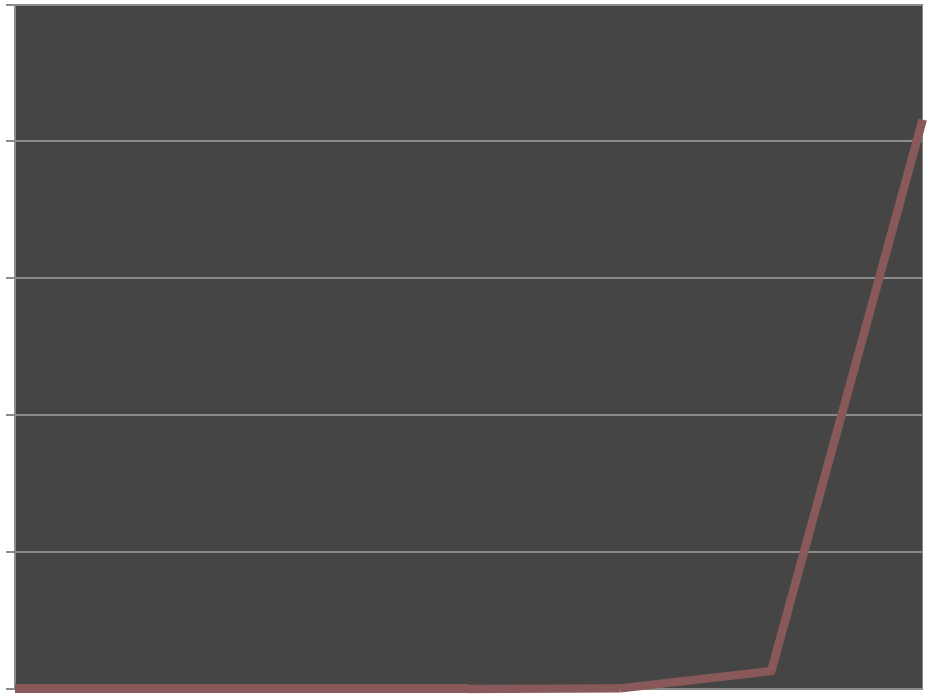
- Un algorithme de reconnaissance naïf est vulnérable au ReDoS quand :
  - L'automate fini non déterministe représentant l'expression régulière contient un choix non déterministe à l'intérieur d'une boucle



Cas de figure I

- Application .NET utilisant la classe RegEx pour effectuer des validations
- Expression régulière à valider : ^(a+)+$
- Valide les chaînes de caractères formées d'une série de a et terminant par X.

Cas de figure I (suite)



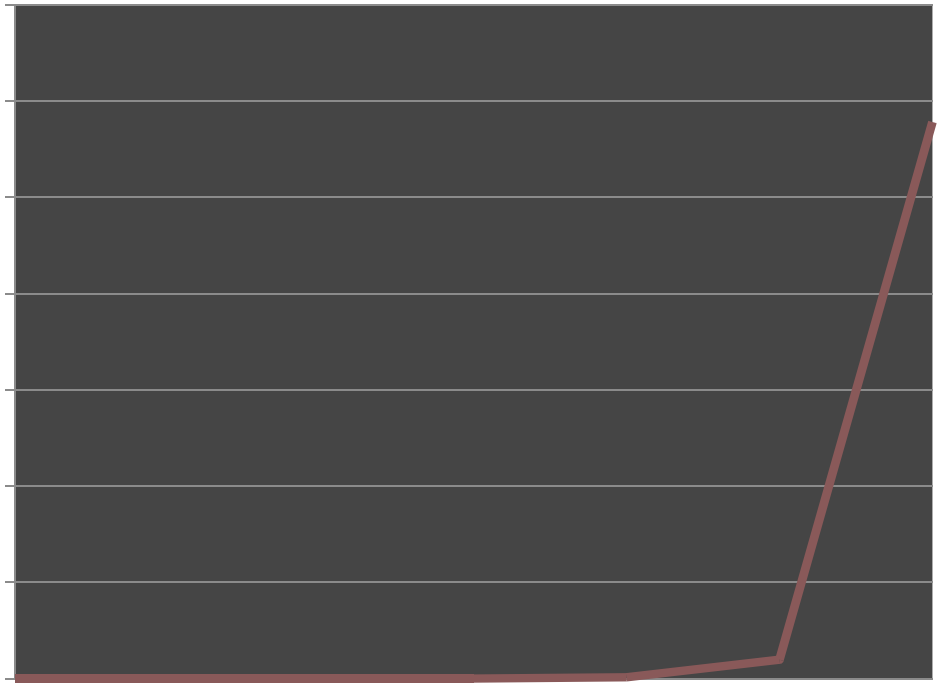
Cas de figure II

- Application faisant la validation de courriel dans un formulaire d'inscription

- Expression régulière à valider : ^([a-zA-Z0-9])**((**[\-.]|[_]+)?([a-zA-Z0-9]+))*(@){1}[a-z0-9]+[.]{1}(([a-z]{2,3})|([a-z]{2,3}[.]{1}[a-z]{2,3}))$

- Valide les chaînes de caractères formées d'une série de a.

Cas de figure II (suite)



Solutions Possibles

Pistes de solutions

- Utilisation de meilleurs algorithmes de validation
- Écrire de meilleures expressions régulières
- Pré-compilation d'expressions régulières

Meilleurs algorithmes de validation

- Pour les expressions régulières pures c'est faisable
- Pour les expressions régulières étendues ce n'est pas faisable
- Peu ou pas de contrôle sur les algorithmes utilisés dans l'industrie
- Solution non-viable

Réécriture d'expressions régulières

- But :
  - Écrire des expressions régulières qui sont traduites en automates fini déterministes par l'algorithme de reconnaissance naïf

- Concept théorique
  - Pour chaque automate fini non déterministe, il existe un automate fini déterministe équivalent

Réécriture d'expressions régulières

- Des règles d'équivalences peuvent être utilisées pour « simplifier » les expression régulières.

$$e+ = ee*$$

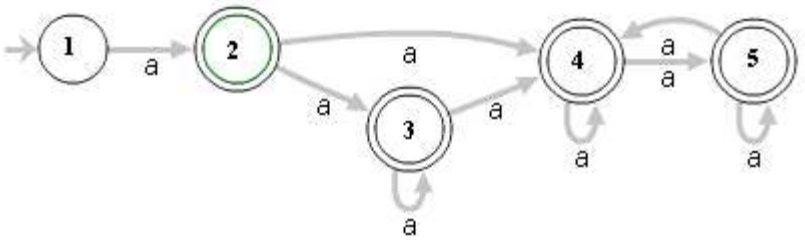
$$e \mid e = e$$

$$e1(e2 \mid e3) = e1e2 \mid e1e3$$

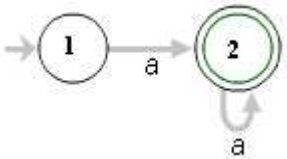
$$(e1 \mid e2)* = e1*(e2e1*)*$$

Cas de figure

^(a+)+$



^a+$



Cas de figure (suite)

Limitations

- Difficile de simplifier en utilisant des règles d'équivalence dans les cas complexes

  (([\-.]|[_]+)?([a-zA-Z0-9]+))*

  - L'expression régulière résultant de la simplification est difficile à lire dans certains cas

    [a-zA-Z0-9]*(([\-.]|[_]+)[a-zA-Z0-9]+)*

  - Difficile de déterminer si l'automate résultant de la traduction est déterministe

Pré-compilation

- Effectuer la conversion en automate fini déterministe avant la validation
- Ne peut être effectué au « Runtime »
- Concept théorique
  - La transformation d'un automate fini non déterministe en automate fini déterministe prend un temps exponentiel

Pré-compilation (suite)

- Précompilation des expressions régulières
  - Utiliser un outil automatisé qui transforme les expressions régulières lisibles en expressions régulières sûres au moment de la compilation

Conclusion

En résumé

- Les expressions régulières sont partout !
- Beaucoup d'outils, d'applications, de FrameWork, etc sont vulnérables aux ReDos
- Les ReDoS sont causés par deux facteurs :
  - Des expressions régulières mal construites
  - Des algorithmes de validations « mal conçus »

Contre-mesures

- Améliorer la maîtrise des expressions régulières afin d'en écrire de meilleurs (non-evil regex)

- Pré-compiler les expressions régulières lorsque cela est possible

Questions ?

Références

ReDoS – OWASP

(https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS)

REGEX – WikiPEDIA

(http://en.wikipedia.org/wiki/Regular_expression)

Mémoire de Maîtrise de Claude Bolduc

(http://archimede.bibl.ulaval.ca/archimede/meta/23728)