Eric Gingras, directeur de la R&D
Sébastien Duquette, analyste en sécurité

Le fuzzing et les tests d'intrusions

Gardien Virtuel
sécurité informatique

Objectifs de la présentation

- Être complète en soit
  - à l'exception des vulnérabilités
- Faire l'état de l'art :
  - du fuzzing dans les tests d'intrusions
  - de l'utilisation du fuzzing dans les outils pour les tests d'intrusions d'applications Web
- Présenter des exemples concrets

Contenu de la présentation

- Le concept de "test d'intrusion"
- Le fuzzing
  - définitions
  - méthodologies
- Les difficultés du fuzzing dans le cadre d'un test d'intrusion
- Le fuzzing et les applications Web
- Démo

CONTENU DE LA PRÉSENTATION

- **Le concept de "test d'intrusion"**
- Le fuzzing
  - définitions
  - méthodologies
- Les difficultés du fuzzing dans le cadre d'un test d'intrusion
- Le fuzzing et les applications Web
- Démo

LA DÉFINITION DE "TEST D'INTRUSION"

« une série d'activités effectuées pour détecter et exploiter les failles de sécurité afin de déterminer dans quelle mesure il est facile ou difficile pour un tiers de déjouer les contrôles de sécurité d'une organisation ou d'accéder sans autorisation à son information et à ses systèmes »[1]

Pour évaluer la sécurité (confidentialité, intégrité et disponibilité) d'une organisation

- robustesse des systèmes
- protections
- politiques et procédures

1 – Source : le Grand Dictionnaire Terminologique de l'OLF (http://www.granddictionnaire.com)

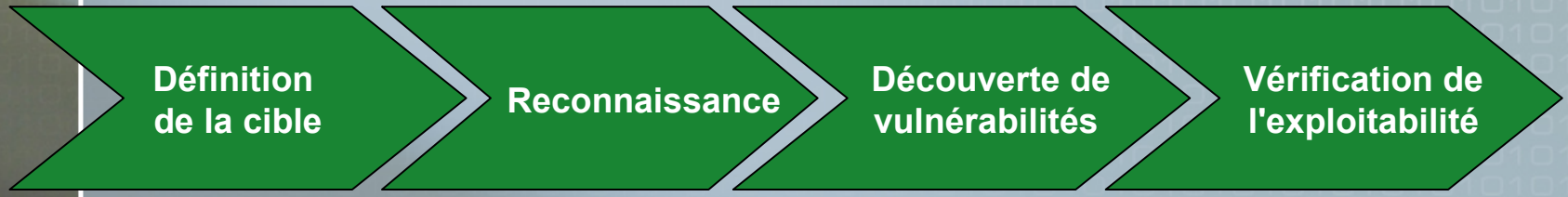
On a besoin de tests d'intrusions

- Parce que les logiciels deviennent de plus en plus complexes :
  - de plus en plus de lignes de code
  - ouverture des systèmes (Internet)
  - nombre de couches
  - logiciels plus puissants
  - expertise en sécurité informationnelle [1]

1 – SANS (http://www.sans.org/top-cyber-security-risks/) : "60% des problèmes de sécurité sur Internet sont liés aux applications Web"

LA MÉTHODOLOGIE DES TESTS D'INTRUSIONS

- Déroulement d'un test d'intrusion :



- Portée d'un test :
  - Test boîte noire (entrées et sorties)
  - Test boîte blanche (code source)
  - Test boîte grise (exécutable)

CONTENU DE LA PRÉSENTATION

- Le concept de "test d'intrusion"
- **Le fuzzing**
  - **définitions**
  - **méthodologies**
- Les difficultés du fuzzing dans le cadre d'un test d'intrusion
- Le fuzzing et les applications Web
- Démo

Définition du "fuzzing"

- Une technique de test qui consiste à injecter des données :
  - invalides
  - inattendues
  - aléatoires
- Dans le but de générer :
  - une opération erronée
  - une exception
  - un résultat inattendu
  - un crash, etc.

LE FUZZING EST UN PARADIGME DE TEST

- Buts des méthodes de tests :
  - **Tests exhaustifs** : trouver toutes les failles
  - **Méthodes formelles** : garantir l'absence de failles (assurance de qualité)
  - **Fuzzing** : trouver des vulnérabilités

LE FUZZING FONCTIONNE

- **2006** : Month of Browser Bugs
  - majorité des bugs découverts par fuzzing
  - publication de fuzzers : AxMan, Hamachi, MangleMe, etc.
- Le fuzzing fait partie du SDL (Security Development Lifecycle) de Microsoft
- Septembre **2009**, découverte d'une vulnérabilité majeure dans SMBv**2** (Windows Vista **& 2008**) par L. Gaffié
  - trouvée en **3** secondes, **15** paquets ...

1 - Source : http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3103

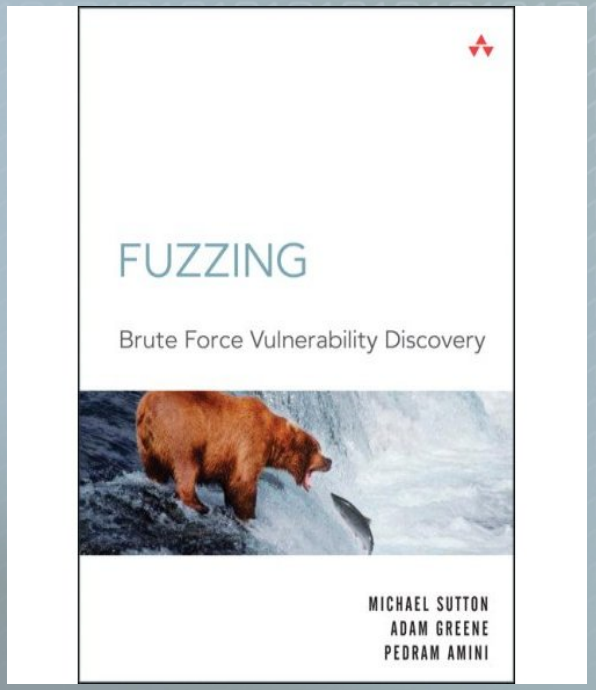
... ET 20 LIGNES DE CODE

```
from socket import *
from time import sleep
from random import choice

host = "IP_ADDR", 445

#Negotiate Protocol Request
packet = [chr(int(a, 16)) for a in """
00 00 00 90 ff 53 4d 42 72 00 00
00 00 18 53 c8 00 00 00 00 00 00 00
00 00 00 00 00 ff ff ff fe 00 00 00 00
00 6d 00 02 50 43 20 4e 45 54 57 4f
52 4b 20 50 52 4f 47 52 41 4d 20 31 2e
30 00 02 4c 41 4e 4d 41 4e 31 2e 30
00 02 57 69 6e 64 6f 77 73 20 66 6f 72
20 57 6f 72 6b 67 72 6f 75 70 73 20 33
2e 31 61 00 02 4c 4d 31 2e 32 58 30 30
32 00 02 4c 41 4e 4d 41 4e 32 2e 31
00 02 4e 54 20 4c 4d 20 30 2e 31 32
00 02 53 4d 42 20 32 2e 30 30 32 00
""".split()]
```

```
while True:
  #/Core#
  what = packet[:]
  where = choice(range(len(packet)))
  which = chr(choice(range(256)))
  what[where] = which
  #/Core#

  #sending stuff @host
  sock = socket()
  sock.connect(host)
  sock.send(' '.join(what))
  sleep(0.1) # dont flood it
  print 'fuzzing param %s' %
     (which.encode("hex"))
  print 'complete packet %s' %
     (''.join(what).encode("hex"))
  sock.close()
```

LA MÉTHODOLOGIE DU FUZZING

1. Identification de la cible
2. Identification des vecteurs d'entrée
3. Génération des données pour le fuzzing
4. Exécution des tests
5. Analyse des exceptions générées
6. Analyse de l'exploitabilité des vulnérabilités découvertes



CIBLES DU FUZZING

- Interpréteurs de commandes
  - arguments d'une commande
- Variables d'environnement
- Fichiers
- Librairies
- Navigateurs web
  - Javascript, ActiveX, Java
  - Plugins : Flash, Adobe Reader, etc.

CIBLES DU FUZZING (SUITE)

- Protocoles réseau
  - Protocoles simples
    - ASCII
    - peu d'authentification
    - échanges linéaires
  - Protocoles complexes
    - binaires
    - sécurisés, contrôle d'erreurs
    - Fragmentés
- Applications Web
  - Vulnérabilités courantes (XSS, CSRF, SQLi, etc.)

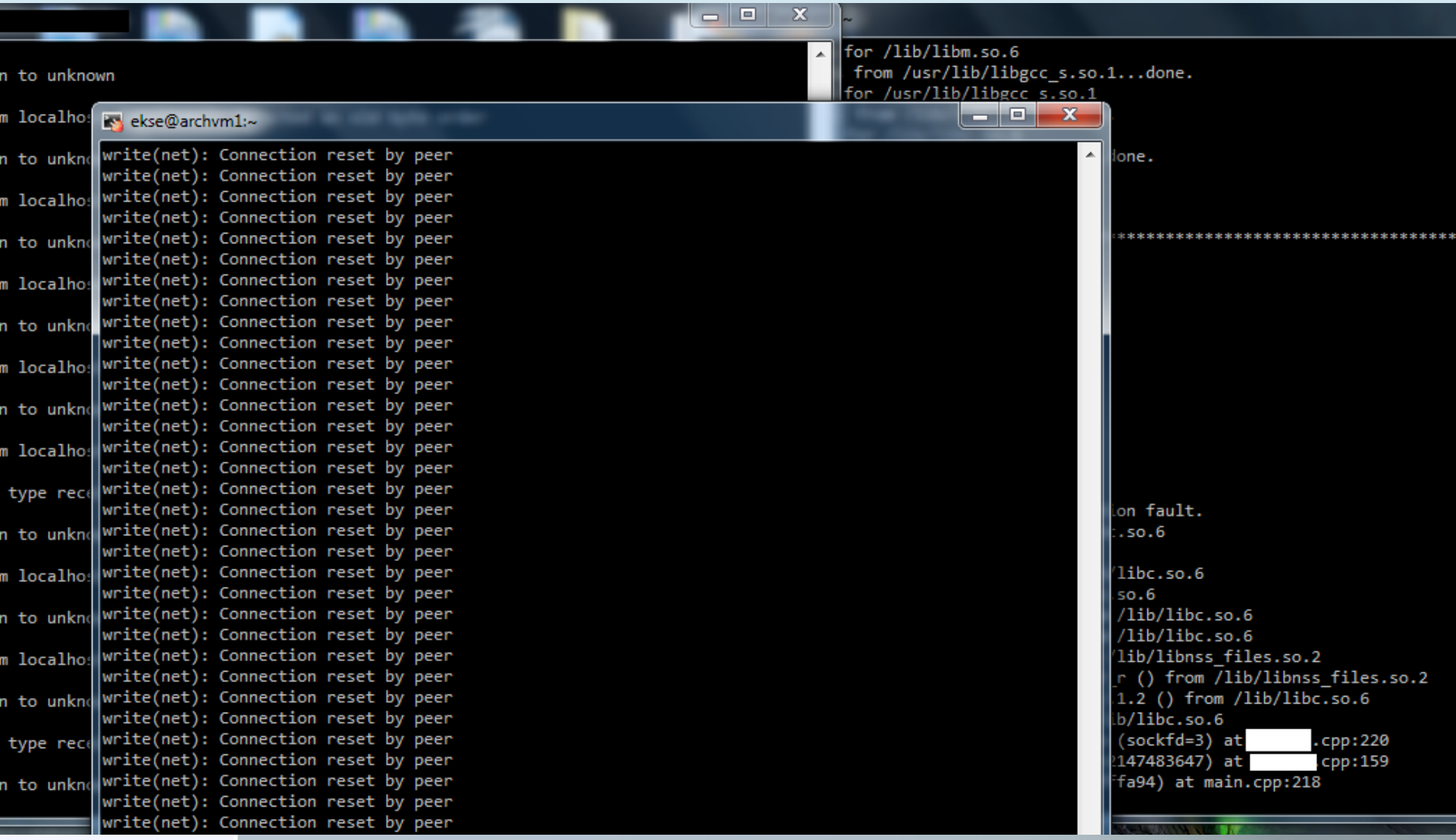
LA GÉNÉRATION DE DONNÉES POUR LE FUZZING

- Par génération
  - Aléatoire
  - Pré-génération
  - Génération de protocoles
- Par mutation
  - Mutation manuelle
  - Mutation automatisée

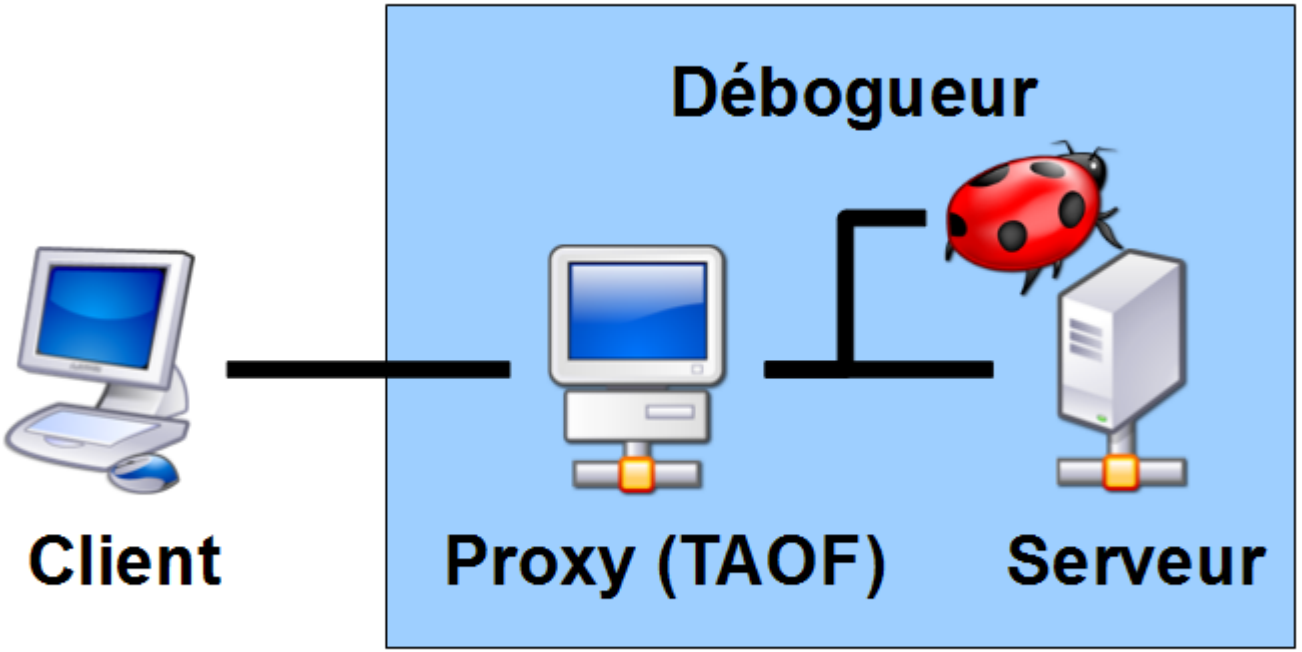
LA GÉNÉRATION DE DONNÉES ALÉATOIRE

“Poor-man fuzzer's”

while true; do cat /dev/urandom | nc ADRESSE_IP PORT; done


ekse@archvm1:~
write(net): Connection reset by peer
for /lib/libm.so.6
from /usr/lib/libgcc_s.so.1...done.
for /usr/lib/libgcc_s.so.1

LA MUTATION DE DONNÉES AUTOMATISÉE



LA GÉNÉRATION DE PROTOCOLE

```
from sulley import *

s_initialize("join")
s_string("NomJoueur")
s_static("^^Ar1111111\n")
s_static("^^Ac")
s_string("NomJoueur")
s_static("\n")

s_initialize("jet")
s_static("^AV")
s_word(1, format="ascii")
s_static("\n")

sess = sessions.session(timeout=5, sleep_time=0.1)
sess.connect(s_get("join"))
sess.connect(s_get("join"), s_get("jet"))

target = sessions.target("127.0.0.1", 7902)
sess.add_target(target)
sess.fuzz()
```

SOURCE DU PROBLÈME

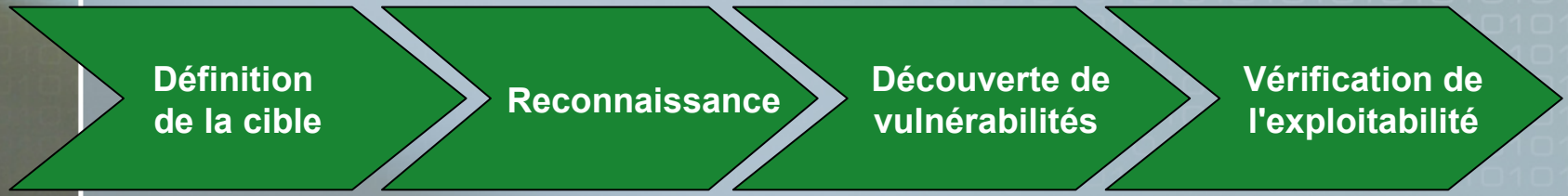
- Lorsqu'un déplacement est demandé, le numéro de ville est utilisé comme index dans un tableau.

```
case C_REQUESTJET:
 i = atoi(Data);
....
else if (i != Play->IsAt && (NumTurns == 0 || Play->Turn
<>EventNum == E_NONE && Play->Health > 0) {
dopelog(4, LF_SERVER, "%s jets to %s", GetPlayerName(Play),
Location[i].Name);
```

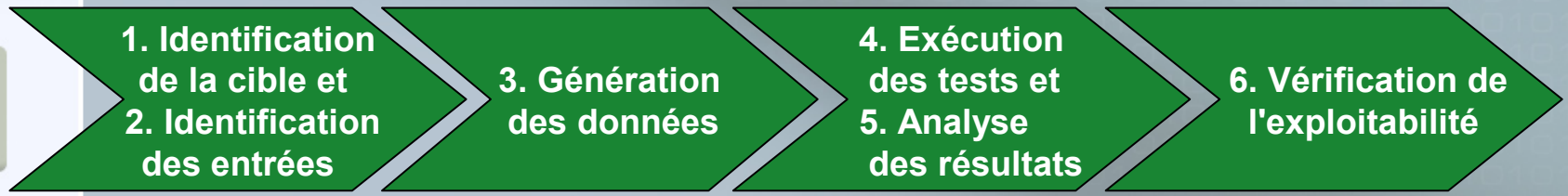
- Bug découvert par dougtko, corrigé dans la version **1.5.13**

Méthodologie des tests d'intrusions vs fuzzing

- Méthodologie des tests d'intrusions



- Méthodologie du fuzzing



Contenu de la présentation

- Le concept de "test d'intrusion"
- Le fuzzing
  - définitions
  - méthodologies
- **Les difficultés du fuzzing dans le cadre d'un test d'intrusion**
- Le fuzzing et les applications Web
- Démo

LES CONTRAINTES LIÉES AUX TESTS D'INTRUSION

- Manque de visibilité
  - signaux invisibles : logs, mémoire, fichiers, etc.
  - Impossibilité d'instrumentaliser le serveur
- Risques de faire planter le serveur
  - « peu recommandé » sur les serveurs de production
- Utilité restreinte
  - recréer environnement de façon locale
    - long, pas toujours possible, faible ROI
  - trouver des dénis de service (DoS)

CONTENU DE LA PRÉSENTATION

- Le concept de "test d'intrusion"
- Le fuzzing
  - définitions
  - méthodologies
- Les difficultés du fuzzing dans le cadre d'un test d'intrusion
- **Le fuzzing et les applications Web**
- Démo

LES APPLICATIONS WEB

- Les applications web représentent une cible intéressante étant donné :
  - La standardisation des protocoles
    - HTTP, HTML, XML, SOAP
  - Applications "maison" : code peu testé, développement ad-hoc
  - Couches élevées, donc moins de risque de crash
  - Messages d'erreurs souvent bavards

IDENTIFICATION DE LA CIBLE ET DES ENTREES

- Entrées d'une application Web
  - Paramètres HTTP GET/POST
  - Formulaires (cas particulier des paramètres POST)
  - Champs cachés
  - Paramètres HTTP : Cookie, Referer, Browser

Génération des données

- Par dictionnaire
  - liste de chaines qui risquent de générer des erreurs si l'application est vulnérable
- Par mutation
  - Encodages : jeux de caractères, entités HTML, double encodages
  - Multiples valeurs pour un même paramètre
  - Valeurs autres que celles permises pour un champ (validation javascript, limite de champ)
  - *Mix and match* : utilisation simultané de valeurs qui ne devraient pas être ensemble

Analyse des résultats et exploitation

- Signaux observables
  - HTTP Status codes
  - Messages dans la page
    - Erreurs PHP, ASP, JSP : demande que les erreurs soient affichées
    - « soft » **(404)**
    - Comparaison avec le message attendu
  - Taille de la réponse
  - Connexions interrompues
- Signaux invisibles : logs, traces du serveur, inspection des données (SQL)

OWASP TOP10 ET LE FUZZING

Classe de vulnérabilité	Utilité du fuzzing
Cross-Site Scripting	Beaucoup
Injection flaws	Beaucoup
Malicious File execution	Un peu
Direct Object Reference	Beaucoup
Cross-Site Request Forgery	Beaucoup *
Information Leakage / Error Handling	Beaucoup
Broken Session Management	Du tout
Insecure Cryptographic Storage	Du tout
Insecure communications	Un peu
Failure to restrict URL access	Beaucoup

Outils disponibles

- Scanneurs utilisant une base de données de vulnérabilités :
  - Nessus, Nikto
  - ne recherchent pas de nouvelles vulnérabilités
- Scanneurs utilisant le modèle « Test-case » :
  - Paros Proxy, w3af, wapiti, XSSploit
  - utilisation de patterns connus.
- Fuzzing manuel :
  - WebScarab, Burp, Paros, w3af
- Commerciaux
  - WebInspect, Acunetix, Rational Appscan

DÉMO

QUESTIONS ?

Démo 1 : regex

- Expressions régulières sont répandues pour la recherche et la validation de paramètres

  ^(z[eé]ro|un|deux|trois|quatre|cinq|six|sept|huit|neuf)$

- devient rapidement très complexe à lire

  ^(((25[0-5]|2[0-4][0-9]|19[0-1]|19[3-9]|18[0-9]|17[0-1]|17[3-9]|1[0-6][0-9]|1[1-9]|[2-9][0-9]|[0-9])\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9]))|(192\.(25[0-5]|2[0-4][0-9]|16[0-7]|169|1[0-5][0-9]|1[7-9][0-9]|[1-9][0-9]|[0-9]))|(172\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|1[0-5]|3[2-9]|[4-9][0-9]|[0-9])))\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])$

Démo 1 : REGS

- Les expressions régulières sont un langage de description formelle
  - Certaines expressions rendent l'évaluation complexe
  - erreurs : répétitions imbriquées, alternances avec chevauchements

"People with little regex experience have surprising skill at coming up with exponentially complex regular expressions."

www.regular-expressions.info

Démo 1 : ReDoS

- Attaque : viser les pire cas d'évaluation d'expressions régulières
  - Contexte des applications web :

    Alex Roichman(Checkmarx) OWASP Israel **2009**

    http://www.owasp.org/images/f/f1/OWASP_IL_**2009**_ReDoS.ppt
- Vecteurs d'entrées
  - Construire une chaîne de caractères pour une expression régulière existante
  - Injecter une expression régulière dans une application qui les construit dynamiquement

DÉMO 1 : REDOS

- Ça arrive même aux meilleurs ! (OWASP validation Project)
- Exemple **1** : nom d'une personne

  Regex : ^[a-zA-Z]+(([\'\,\.\-][a-zA-Z])?[a-zA-Z]*)*$

  ReDoS : "aaaaaaaaaaaaaaaaaaaaaaaaaaaaa!"
- Exemple **2** : nom d'une classe Java

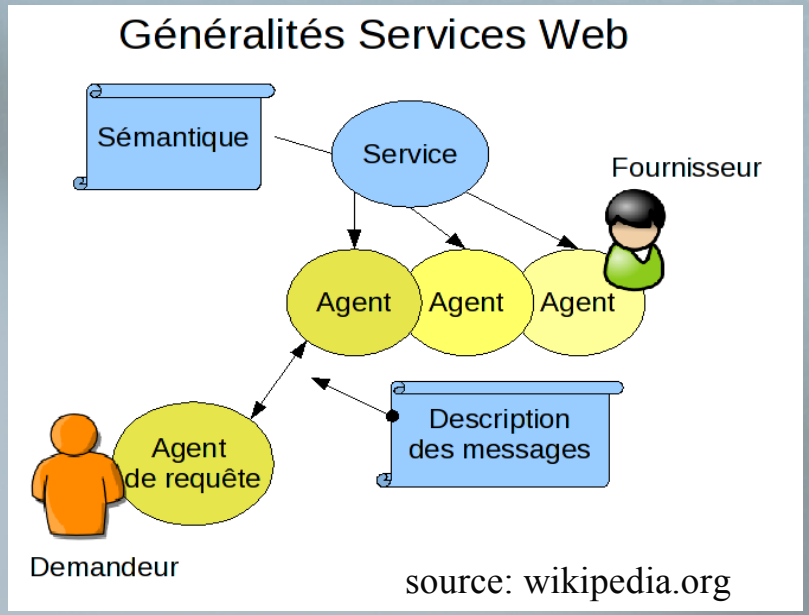
  Regex : ^(([a-z])+.)+[A-Z]([a-z])+$

  ReDoS : "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!"

1 – « Developers implementing input validation engines should review these regular expressions. Save the time of developing a complicated regular expression that currently exists! ». [http://www.owasp.org/index.php/Category:OWASP_Validation_Project]

DÉMO 2 : SERVICES WEB

Selon Wikipédia : *"un programme informatique permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués."*



DÉMO 2 : SERVICES WEB

Généralement 3 composants :

- Protocole de transport(SOAP, XML-RPC, REST)
  - définit la structure des requêtes
- Langage de description(WSDL, WADL)
  - définit les appels rendus disponibles par le service (noms des méthodes, types des variables)
- Langage de sérialisation des données(XML, JSON)
  - définit la représentation des données échangées

DÉMO 2 : SERVICES WEB

- La description permet de connaître la forme des appels par analyse syntaxique : idéal pour automatisation

```
<wsdl:operation name="GetLastTradePrice">
   <wsdl:input message="tns:GetLastTradePriceInput"/>
   <wsdl:output message="tns:GetLastTradePriceOutput"/>
 </wsdl:operation>

 <wsdl:message name="GetLastTradePriceOutput">
   <wsdl:part name="body" element="xsd1:TradePrice"/>
</wsdl:message>

<xsd:element name="TradePrice">
 xsd:<complexType>
       <xsd:all>
            <xsd:element name="price" type="float"/>
       </xsd:all>
 </xsd:complexType>
</xsd:element>
```

nom de l'appel

Types

nom des variables

source : www.w3.org

DÉMO 2 : SERVICES WEB

Tests possibles :

- *Analyse syntaxique* des requêtes
  - requêtes mal-formées, paramètres redondants, etc.
- Code de l'application :
  - tests ciblés en fonction des types
  - même failles que pour les applications web

DÉMO 2 : SERVICES WEB

QUESTIONS ?