1 - Introduction à la cryptologie

cryptologie = cryptanalyse + cryptographie
cryptographie : techniques mathématiques liées aux aspects (intégrité, confidentialité,etc…) de la sécurité de l'information.
cryptanalyse : découvrir les secrets, décrypter, se déguiser

Confidentialité: protection contre les accès non autorisés à l'information
Intégrité: protection contre les modifications non autorisées de l'information
Authentification (des données et des entités):

- Auth des données: authenticité de l'origine,du contenu, de validité → intégrité
- Auth des entités: identification des entités d'une communication

Non-répudiation: empêcher une entité de nier une action passée

E primitive de chiffrement, D=E^{-1} primitive de déchiffrement : E cohérente SSI E bijective, et $\forall m \in M, E(D(m)) = m$

Fonction à sens unique : fonction f simple à calculer, dont l'inverse est impossible à calculer pour la majorité des images $y \in Im(f)$

- Fonction à sens unique avec trappe: fonction à sens unique avec propriété supplémentaire: étant donnée une information appelée *trappe*, il est possible $\forall y \in Im(f)$de trouver $x \in X / f(x) = y$

L'existence de ces fonctions n'est pas établie, néanmoins de bons candidats sont utilisés dans les schémas cryptographiques actuels. Les fonction à sens unique sont essentielles pour la cryptographie à clé publique

Permutation: bijection de S dans S : $p : S \to S$, souvent utilisées dans les schémas cryptographiques
Involution : bijection de S dans S telle que $f = f^{-1}$ ($f(f(x)) = x$), souvent utilisées dans les schémas crypto.

Principe de Kerckhoffs: la sécurité d'un système de chiffrement ne doit reposer que sur le secret d'une clé, qui doit être facilement modifiable (erreurs du passé : César/Vigenère/Scytale/Enigma…)

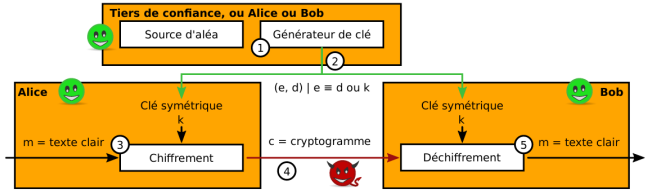
Alphabet de définition: ensemble fini A
Ensemble de départ: les clairs, ensemble des messages M, chaînes d'un alphabet A
Ensemble d'arrivée: les chiffrés, ensemble des chiffrés C, chaînes d'un alphabet A

ensemble des clés noté K,$e \in K$est une clé (de chiffrement ou déchiffrement)
Fonction de chiffrement : chiffrer un message, $e \in K$ détermine **une seule bijection**$E_e : M \to C$
Fonction de déchiffrement : déchiffrer un chiffré, $d \in K$détermine **une seule bijection**$D_d : C \to M$
Schéma de chiffrement :

- Ensemble de transformations de chiffrement $E_e | e \in K$
- Ensemble de transformations de déchiffrement $D_d | d \in K$
- M et C sélectionné avec les propriétés suivantes:
  - $\forall e \in K \exists$une unique clé $d \in K / D_d = E_e^{-1}$
  - (e,d) = paire de clés, e peut être égal à d

Algo de chiffrement cassé : un attaquant qui ne connaît pas (e,d) peut retrouver le clair associé à un chiffré en un temps inférieur à la force brute
Force brute : parcourir K à la recherche de d permet de casser un schéma en une durée donnée → |K| doit être choisi pour rendre cela infaisable
Service = méthode pour assurer des propriétés de sécurité, attaque = casser un objectif de sécurité.

Chiffrement Symétrique



Facile de déterminer d à partir de e → souvent d=e
Schéma de chiffrement par bloc: découpe les clairs en blocs de taille t, qui sont chiffrés les uns après les autres souvent par substitution ou transposition

Problème: il faut distribuer la clé sur un canal sécurisé pour pouvoir sécuriser un canal

substitution monoalphabétique:
$E_e(m) = (e(m_1)e(m_2)\dots e(m_t)) = (c_1c_2\dots c_t) = c$ avec $m = (m_1m_2\dots m_t)$, conserve la fréquence d'apparition des lettres
→ on peut inverser la transformation en observant assez de chiffrés : chaque permutation $p_i$se répète modulo t. Si on connaît t, on peut refaire une analyse fréquentielle sur chaque groupe de chiffrés modulo t et retrouver $p_i$avec une faible quantité de chiffrés

Chiffrement par transposition: permute les symboles dans un bloc de taille t, produit donc des anagrammes.

Chiffrement de César (I siècle av JC)

On décale chaque lettre d'un certain nombre de lettres (fixe)

Craquable avec analyse fréquentielle

Très facile car deux même lettres (e par exemple) sont chiffrées de la même manière

Chiffrement de Vigenère (XVIe siècle)

Une lettre de l'alphabet peut être chiffrée de plusieurs manières.

Clé = une chaîne de caractères. Un caractère de la clé = un décalage qui correspond à la position de la lettre dans l'alphabet

Comment le cracker ?

On essaye de chercher des motifs qui se répètent dans le chiffré et on note leur séparation en termes d'index. On calcule le pgcd des séparations

Signatures

Associe l'identité d'une entité à de l'information

Signer = transformer $m \in M$et une information secrète en un tag $s \in S$appelé signature
Signatures essentielles pour les services:

- Authentification des entités
- Non-répudiation

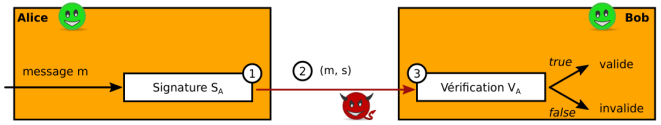
Transformation de signatureS_A: $M \rightarrow S$gardée secrète par A

Transformation de vérif de signature: V_A: $M\,x\,S \rightarrow \{true, false\}$, publique

Kerckhoff → usage de clés pour $V_A$et S_A

Schéma de signature:

- transformation de signature $S_A$caractérisée par une clé $k_A$secrète
- transformation de vérification de signatures de A V_A caractérisée par une clé $I_A$publique



Authentification

Scénario 1: entités communiquant sur un réseau, vérification de leurs identités (éventuellement dans les 2 sens)
Exemple : HTTPS (TLS), SSH (SSH-USERAUTH, SSH-TRANS, IPSEC/IKE,...
Scénario 2 :

▶ Vérification de l'origine des données en temps différé

Protocole possible : A envoie au serveur S un message avec sa signature $S_A(m)$. B Récupère le message depuis S et vérifie l'origine du message à l'aide de V_A.

A S B

$s = S_A(m)$

$\xrightarrow{(s,m)}$

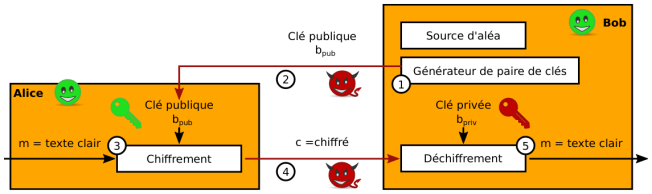
$\xrightarrow{(s,m)}$

authentifié $= V_A(m, s)$

(PGP, SMIME)

Chiffrement à clé publique

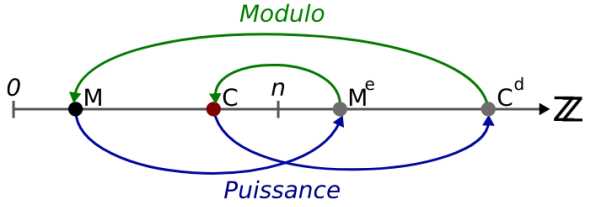
$\forall(E_e, D_d)$,impossible de trouver $m \in M / E_e(m) = c \rightarrow$ impossible d'inverser la transformation de chiffrement $E_e$est une fonction à sens unique avec trappe.
e = clé publique, d = clé privée



Exemple: RSA (Rivest, Shamir, Adelman)

Création des clés:

- choisir p et q premiers distincts, calculer n = pq et $\phi(n) = (p-1) * (q-1)$indicatrice d'Euler
- choisir e premier avec $\phi(n)$, calculer d : $e * d \equiv 1\, mod(\phi(n))$ (algo d'Euclide étendu)



Avantages:

- pas de confiance mutuelle entre émetteur/récepteur
- Gestion de clé "facile" : répertoire public de clés publiques et clés privées jamais transmises
- Nouvelles utilisations : distribution de clés symétriques, signatures, certificats...

⚠ RSA est malléable :

Alice envoie à Bob c_1 sur un canal non sécurisé tel que :

$$m_1^e \equiv c_1 \pmod{n}$$

Eve en homme dans le milieu transforme c_1 en c tel que :

$$c = c_1 c_2 \mid c_2 \equiv m_2^e \pmod{n} \quad m_2 \text{ inconnu d'Eve}$$

Bob reçoit c et déchiffre m' tel que :

$$m' \equiv c^d \equiv (c_1 c_2)^d \equiv c_1^d c_2^d \equiv m_1^{ed} m_2^{ed} \equiv m_1 m_2 \pmod{n}$$

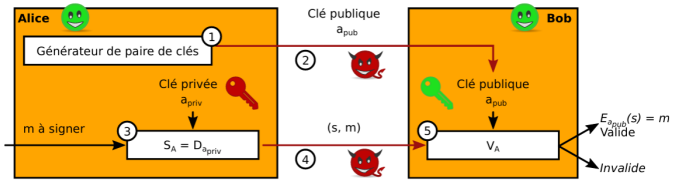
Par associativité de la multiplication et RSA

⚠ Attaque par homme du milieu:
Si Eve intercepte l'échange des clés publiques, elle peut remplacer la clé publique de Bob par la sienne aux yeux d'Alice et inversement, lui permettant de déchiffrer, lire puis rechiffrer les messages. Eve se fait passer pour Alice aux yeux de Bob et pour Bob aux yeux d'Alice

Schéma de signature avec chiffrement à clé publique

Signature: $m \in M$: $s = D_{a_{priv}}(m)$

Vérification: $V_A(m, s) = valide\ si\ E_{a_{pub}}(s) = m \mid invalide\ sinon$



Problèmes: le produit de 2 signatures est une signature du produit des deux clairs, et $V_A$donc $E_e$étant publiques, l'adversaire peut choisir au hasard une signature et calculer le message associé, $m_{forgé} = E_e(s_{forgé})$ →forge de signatures, car $V_A(m_{forgé}, s_{forgé}) = valide$

Solution: Choisir un ensemble de messages signables $M' \subset M$ $et\ |M'| \ll |M|$.
Pour choisir $s_{forgé}$valide tel que $E_e(s_{forgé}) \in M'$, l'adversaire doit inverser E_e →impossible.

En pratique, on utilise des fonctions à sens unique (fonctions de hashage par ex) pour calculer $m' \in M'$avec des m de taille arbitraire

Fonctions de Hashage

Fonction associant des chaînes binaires de taille arbitraire à des chaînes de taille t fixe.

$P[C = c|M = m] = 2^{-n}$

Résistance aux préimages: impossible de calculer m tel que $H(m) = c$

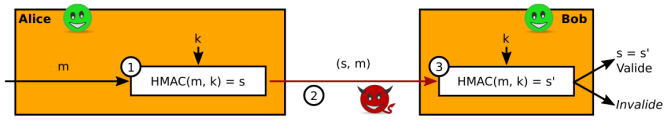
Résistance aux secondes préimages : connaissant $(m, c)/\ H(m) = c$,impossible de calculer $m' \neq m\ /\ H(m') = c$

Résistance aux collisions: impossible de calculer $m'\ et\ m\ /\ H(m) = H(m')$

Paradoxe des anniversaires:

$P[trouver\ m\ et\ m'|H(m) = H(m')\ après\ 2^{n/2}\ essais] \geq \frac{1}{2}$

Intégrité



2 - Sécurité des applications et débordements de tampon mémoire

Architecture et jeu d'instruction x86

Le processeur dispose d'un jeu d'instructions étendu (CISC), il est microcodé (pour corriger des bugs et des vulnérabilités) et utilise 3 niveaux de caches. Il y a plusieurs niveaux de privilèges allant de 0 (mode noyau) à 3 (mode utilisateur), certaines instructions ne sont exécutables qu'en mode noyau. Le noyau met en place une virtualisation de la mémoire, il gère les accès en lecture, écriture et exécution, et traduit les adresses virtuelles en adresses physiques.

Registres

Pointeur d'instruction : ip
Généraux : a, b, c, d, r[8-15]
Tête de pile : sp ; base de pile : bp
Index, copie mémoire : si, di
Drapeaux d'état [er]flags : c, z, o, …

Unités mémoire

octet (b) : 8-bit, [abcd][lh] (l poid faible et h poid fort)
mot (w) : 16-bit, [abcd]x, sp, bp, di, ...
double mot (l) : 32-bit, e[abcd]x, esp, ebp, edi, ...
quadruple mot (q) : 64-bit, r[abcd]x, rsp, rbp, rdi, …

Instructions

mov : accès direct ou indirect à la mémoire, affectation de registres
push : un élément est ajouté en sommet de pile et esp est décrémenté
pop : un élément est retiré du sommet de pile et esp est incrémenté
movs : copie directe mémoire (≈ memcpy, @si++ vers @di++)
and, or, xor, test : opérateurs bit à bit
add, mul, sub, div : arithmétique
shl, shr : décalage à gauche et à droite
int : interruption logicielle
syscall : appel système
jmp: saut inconditionnel
je/jne adresse : saut conditionnel
call adresse : appel de fonction (empile l'adresse de retour et saute à l'adresse de la fonction)
ret : saut inconditionnel à l'adresse de retour et la dépile

Exemples d'adressage

Base : rega = mem[regb]
Base + déplacement + index × taille :
rega = mem[regb + d + regi × s]
mov d(regb, regi, s), %r9

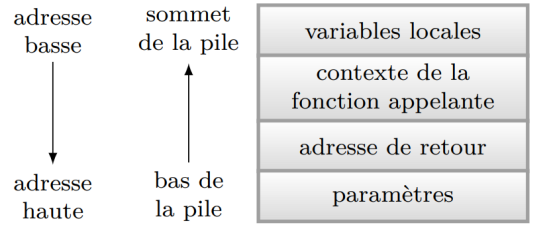
On peut ajouter aux instructions la taille des opérandes en ajoutant un suffixe [bwlq] (ex: movs**b**, pop**q**). Pour la syntaxe AT&T on écrit d'abord la source puis la destination (movl %eax, %esi).

Rappels sur les appels de fonction

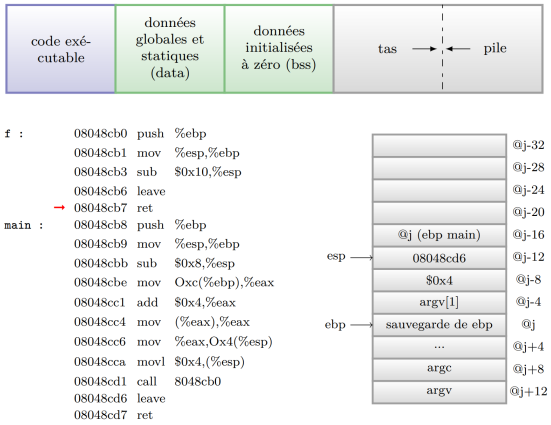
Lors de l'appel d'une fonction en langage C, on empile dans l'ordre :

- Les paramètres de la fonction
- L'adresse de retour (de l'instruction qui suit l'invocation)
- Une sauvegarde du contexte d'exécution de la fonction appelante
- Les variables locales

Lors du retour, on dépile dans l'autre sens.

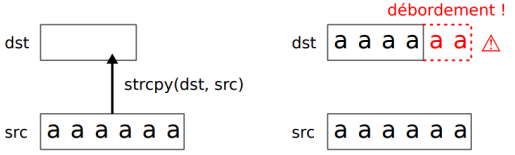


esp est le sommet de la pile, **ebp** est le pointeur de base de pile (lors de l'appel d'une fonction, ce registre doit donc être sauvegardé pour permettre à la fonction appelée de disposer de sa propre zone de pile). La pile grandit vers le bas alors que le tas grandit vers le haut.



Les attaques de type buffer overflow

La majorité des attaques sur les applications proviennent d'erreurs de programmation : entrées non vérifiées, pas de contrôle des appels de programmes externes, utilisation de fonctions non sécurisées (`strcpy`, `strcat`).

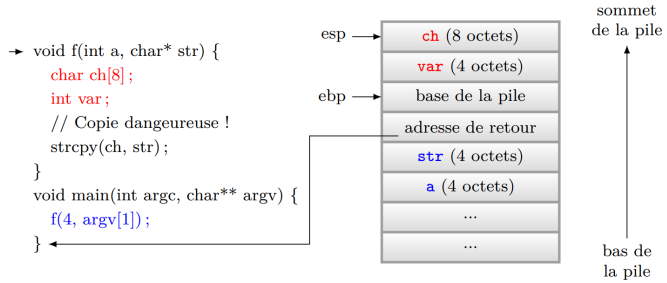


L'enjeu du buffer overflow est de savoir sur quoi on déborde, de trouver comment le modifier de manière cohérente et de connaître la taille maximale du débordement. Un attaquant peut exploiter un buffer overflow pour détourner l'exécution du programme voire lui faire exécuter un code arbitraire. Il est plus intéressant de trouver une exploitation sur un programme exécuté avec les privilèges `root` pour obtenir le contrôle de la machine.

Les possibilités dépendent du lieu du débordement :

- pile : données de contrôle d'exécution ou variables du programme
- tas : données gérées par la libc ou variables du programme
- data : variables du programme
- code : pas accessible en écriture

Pour analyser un programme, on recherche d'abord des fonctions vulnérables comme `strcpy(ch, str)` copie le contenu de `str` à l'adresse `ch` sans vérifier que la place mémoire est suffisante.

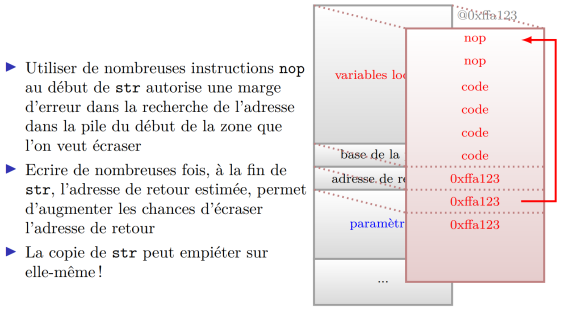


Si la taille de str est supérieure à 8 et inférieure à 12, problème écrasement de `ch` **et des octets de** `var`. Si la taille de `str` est supérieure à 16, problème écrasement de `ch`, de `var`, du pointeur de base de la pile et de l'adresse de retour.

Principe : écraser l'adresse de retour de façon à la remplacer par l'adresse d'une zone mémoire que l'attaquant maîtrise (celle du début de la zone écrasée). Dans cette zone, on a en fait une copie de str, on va donc fabriquer `str` en positionnant le code à exécuter au début du buffer et l'adresse de retour souhaitée à la fin du buffer.

Il reste encore à identifier la position dans la pile de l'adresse de retour pour pouvoir l'écraser et l'adresse dans la pile de la zone mémoire que l'on veut écraser. On peut utiliser gdb avec un breakpoint bien positionné : l'adresse de retour est dans `saved eip` de `info frame`.

Quelques astuces :



Si on suppose que la pile est toujours stockée à la même adresse dans un processus en cours d'exécution, on peut trouver l'adresse de `ch` par essais successifs :
- On détermine l'adresse de base de la pile
- On calcul l'adresse de `str` en soustrayant au moins la taille de `str` plus un offset que l'on fait varier en effectuant plusieurs tests.

Shellcode

Un shellcode doit être court (pour éviter les segfault), ne pas avoir d'octet 0 (fin de chaîne de caractère), être indépendant de la position en mémoire et de la vulnérabilité exploitée, et dépendant du code assembleur.

Récupération du numéro d'un appel système :
`more /usr/include/asm-i386/unistd.h | grep write`
Paramètres des appels systèmes :

- eax → numéro de l'appel système
- ebx → premier argument
- ecx → deuxième argument
- edx → troisième argument

Mécanismes de défense

Systèmes de détection d'intrusion (IDS) : pour les vulnérabilités exploitées à distance, l'attaquant doit envoyer son payload à travers le réseau. L'IDS peut analyser les paquets à la recherche de ces payload et les bloquer.

Randomisation de l'espace d'adresse (ASLR) : L'adresse de la pile dans l'espace d'adressage d'un processus change à chaque exécution.

Pile non exécutable : lors de l'exécution d'un bout de code, le processeur vérifie que la page contenant le code est exécutable en analysant le descripteur correspondant. Sur les noyaux Linux récents, la pile est non exécutable et il est donc impossible de réaliser un buffer overflow dans la pile simplement.

Réordonnancement des variables et des pointeurs (protection de la pile) : les variables et pointeurs ne sont pas placées dans le même ordre que le code source, ce qui rend l'exploitation de buffer overflow compliqué.

Les canaries (protection de l'adresse de retour) :

Au début de la fonction appelée, une valeur est insérée entre les variables locales et la sauvegarde du registre `ebp`.

Un débordement d'une variable locale va écraser cette valeur avant d'écraser le pointeur de retour.

Avant le retour de la fonction, la valeur du canary est vérifiée et si il a été modifié, le programme est arrêté brutalement.

Détection automatique des buffer overflow

Analyse dynamique : Injection de code autour des déclarations de tableaux, remplacement des fonctions de manipulation des tableaux et vérification du comportement à l'exécution.

+ Taux de faux positifs faible
- Pénalité à l'exécution

Analyse statique : Les programmes ne sont pas exécutés mais ils sont analysés par des outils (algorithme Gen/Kill, graphe de contrôle de flux - uno, interprétation abstraite - boon).

3 - Sécurité des réseaux IP

Attaques

Les attaquants:

- personne extérieure → intrusion réseau
- personne intérieure → augmentation des privilèges
- administrateur → abus d'autorité

Ce sont souvent des utilisateurs authentifiés et autorisés
Pourquoi ? Challenge intellectuel, vandalisme…

Principales attaques :

- Écoute passive → on ne modifie pas l'information (*sniffing*, scan réseau, *wiretapping* (boucle locale DSLAM))
- Interception → information modifiée, créée ou détruite (rejeu, insertion, substitution, suppression, vol de session TCP)
- Déguisement → l'attaquant se fait passer pour quelqu'un d'autre (Spoofing IP, MAC, phishing, harponnage)
- Cryptanalyse → récupération d'informations secrètes à partir publique → déchiffrement de message, récupération de clé de (dé)chiffrement…
- Déni de service (distribué) → empêcher la victime d'accéder à un service (spamming,flooding,smurfing)
- Logiciels malveillants / bombes logiques réseau → Portes dérobées, chevaux de troie, protocoles volontairement faibles

Problèmes principaux d'IP :

- Pas de confiance dans le champ IP source
- Les routeurs peuvent modifier le contenu des paquets
- Les routes annonc´ees par les protocoles de routage ne sont pas authentifiées

Couche physique

- Couper les bus et insérer un équipement pour maintenir le lien
- Insertion sur routeurs, hôtes et commutateur réseau (accès physique nécessaire aux serveurs ou de switchs)
- Canaux auxiliaires (émission électromagnétique des câbles (TEMPEST), consommation électrique des équipements)

Couche liaison de données

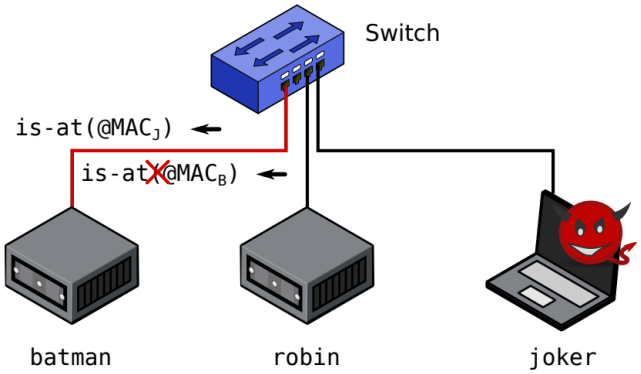
<u>Sniffing</u>
Dans un réseau local avec un hub, l'attaquant se branche avec un contrôleur réseau en mode *promiscuous* → il peut écouter tout le trafic échangé par les hôtes A et B

⇒Switch réseau, pour isoler les communications entre hôtes mais **surtout** diminuer la charge (pas suffisant pour contrer les attaques ARP ou à débordement de tables CAM)

L'écoute réseau est passive → difficile à détecter
⇒ détection par cohérence du protocole :

- Machine M_A écoute, M_V → echo-request(@IP_A,@MAC_X)
- Si M_V ← echo-reply(@IP_M,@MAC_A)
- ⇒ mode *promiscuous* sur M_A

<u>ARP Spoofing</u>
Chaque paquet IP est encapsulé dans une trame Ethernet(@MACdst, @MACsrc). Le protocole ARP sert à résoudre l'adresse IP d'une machine en adresse MAC.



batman → who-has (@IP_r) → switch
switch → who-has (@IP_r) → robin, joker

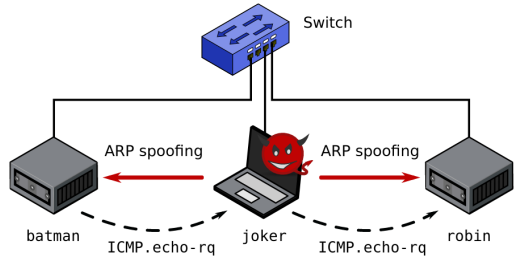
robin → is-at(@MAC_r) → switch
joker → is-at(@MAC_j) → switch

Si la réponse de joker arrive en premier :
switch → is-at(@MAC_j) → batman

Le trafic batman → robin ira maintenant vers joker.
Situation de compétition avec la réponse légitime → inondation
Cache ARP de batman corrompu (ARP cache poisoning)
Attaque à renouveler lors de l'expiration du cache → inondation

<u>Man-in-the-middle ARP :</u> vol du trafic

1. ARP spoofing batman ↔ robin
   - Cache ARP batman : (IP_{robin}, MAC_{joker})
   - Cache ARP robin : (IP_{batman},MAC_{joker})
2. Joker active le relayage des paquets
3. Configuration + fine de la pile IP pour + de furtivité



<u>Inondation ARP</u>

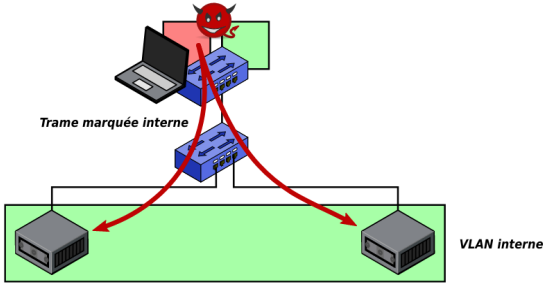
Un switch isole le trafic unicast → commute les trames sur le port concerné
Il tient donc une table de couples (MAC,port), qui peut être remplie jusqu'à déborder → retour au comportement HUB

Saut de VLAN (VLAN *hopping*)
Rappels : Ajout du TAG par le switch d'émission lors de la commutation sur une interface en mode trunk. Retrait du TAG et lecture du VLAN ID lors de la réception sur le switch de réception. Commutation de la trame sur le port si le VLAN ID sauvé correspond au VLAN ID du numéro de port cible.

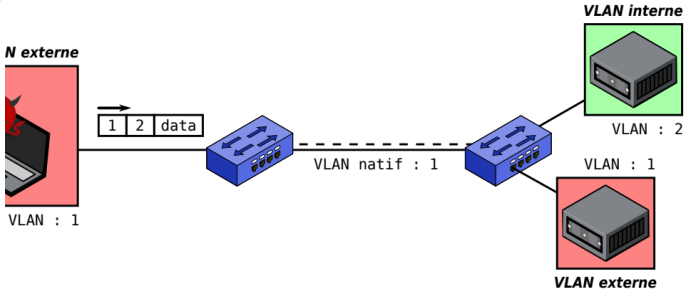
Le but de l'attaque est d'envoyer un message d'un VLAN sur un autre VLAN sur lequel on est pas connecté. Deux méthodes :

Déguisement en switch (DTP)
Sur certains switch, protocole d'auto négociation d'un port (normal ou *trunk=multi-vlan*), notamment avec le protocole propriétaire *Dynamic Trunk Protocol* de Cisco.
Problème de sécurité: pas d'authentification de la classe de l'équipement
Méthode:

1. Se brancher sur le port d'un switch / VLAN non privilégié
2. Négocier ce port en mode *trunk* avec DTP
3. Émettre des trames marquées avec le VLAN cible



Tag double
Nécessite que le VLAN extérieur soit sur le VLAN natif des interfaces *trunk.*



Le tag 1 est enlevé par le switch 1 et le paquet est transmis : switch 1 → 2 | data → switch 2

Le deuxième switch enlève le tag 2 et transmet le message sur le VLAN 2.

Protection simple : ne jamais placer un port d'accès sur le VLAN natif de ports *trunk*.

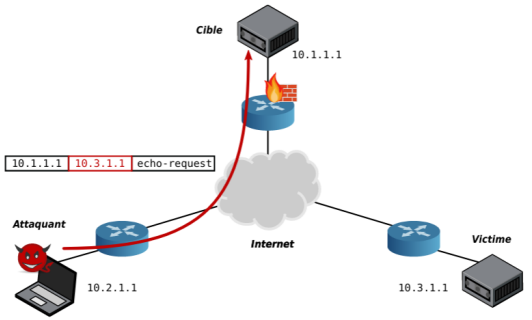
Autres attaques
Protocole Cisco Discovery Protocol (CDP)

- Équivalent IEEE Link Layer Discovery Protocol (LLDP)
- Découvrir les équipements d'un même domaine de diffusion
- Peut remplacer les protocoles de routages (ex IPv6 NDP)
- **Injection de messages** → modification de la vision du réseau
- **Inondation de messages** → déni de service

Protocole STP → Forcer la réélection du switch racine → déni de service

Couche réseau

Déguisement IP
Émettre un paquet avec une IP source différente de la nôtre



⇒ Contournement de filtrage réseau

Déguisement de l'IP de destination
Attaque historique :

- Source routing : impose un routage strict d'un paquet
- Loose source routing : sous-ensemble des routeurs traversés

Permet de contourner un pare-feu :

- Trouver une machine ou un routeur intermédiaire accessible dans le réseau cible
- Ajouter ce routeur en ip de destination IP
- Envoyer le paquet qui est accepté par le pare feu
- Ajouter la cible réelle comme destination réelle dans la liste des IP loose source routing
- Le routeur intermédiaire va permuter l'ip de la cible réelle avec celle du routeur intermédiaire et relayer le paquet à la cible réelle

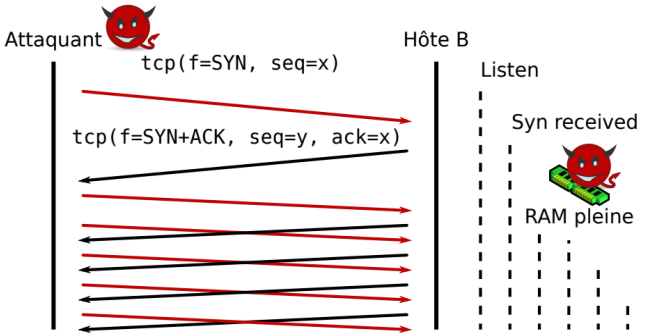
Smurfing
Dénis de service par inondation réseau :

- Envoyer une grosse quantité de requêtes icmp-echo vers l'adresse de diffusion (broadcast) d'un réseau (multiplicateur) et déguiser l'adresse source avec celle d'une machine victime
- Toutes les machines IP du réseau multiplicateur vont répondre à la victime → grosse génération de trafic

Couche transport

Dénis de service TCP
Inondation TCP SYN : ouverture de connexions non fermés (cf. attaque de Kevin Mitnick) :

- L'attaquant génère un certain nombre d'ouverture de connexions
- La victime accepte la connexion et exécute la connexion inverse → consommation mémoire pour l'attente → dénis de service

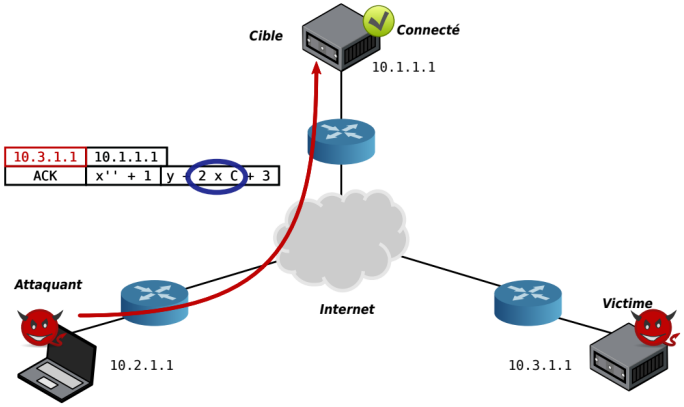


Déguisement TCP / IP

Principe : déguisement IP en devinant les numéros de séquence TCP choisis par la cible.

Cette attaque ne peut fonctionner que sur des systèmes cibles où les numéros de séquences choisis sont prédictibles.

Protection contre cette attaque : randomisation des numéros de séquence utilisés.



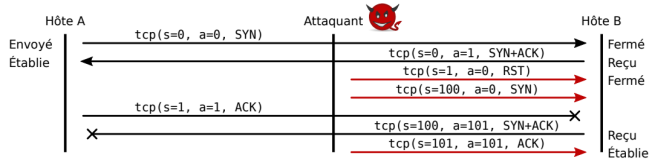
Dénis de service de la victime pour envoyer les paquets à sa place en prédisant le numéro de séquence.

Permet d'obtenir une connexion TCP avec l'IP source de la victime.

Interruption de session TCP

Envoi d'un segment TCP RST portant le bon numéro de séquence (requiert un déguisement IP, triviale si placé en homme dans le milieu, sinon, attaque par force brute).

Désynchronisation de session TCP (lors de la poignée de main)



Les hôtes A et B ont une session TCP désynchronisée (A veut envoyer le paquet 1 alors que B attend le paquet 101), ils ne peuvent pas s'échanger de segments.

Filtrage

Introduction

+d'attaques réseau ⇒ construction d'architectures sécurité réseau:

- définition de domaines aux ≠ niveaux de privilèges
- filtrage des communications inter domaines

Objectif d'une archi sécu réseau:

- Contrôler les flux entrants/sortants des domaines
- limiter la visibilité de la structure réseau

→ Conception du pare-feu

Principe

- Contrôler les flux réseau avec des fonctions de filtrage et/ou traduction
- Habituellement le réseau interne est de confiance, le réseau externe non (ex. Internet)

Objectifs

- Configurable avec un "langage" du trafic légitime
- Seul le trafic légitime est relayé
- Analyse des flux inter domaines → positionnement stratégique
- limiter l'impact sur les perf → matériel dédié et filtrage au bon niveau

Catégories

- Pare-feu (packet filter): routeur de sécu qui filtre le trafic aux niveaux réseau et transport
- Serveur mandataire (proxy): peuvent souvent filtrer jusqu'à la couche application. Fonctionnement explicite (classique) ou transparent

Filtrage réseau

- Contrôle de flux en fonction de règles de filtrage
- Exécuté par du matériel spécifique ou des machines généralistes configurées
- Règles s'appliquant au moins aux champs des protocoles réseau et transport (par ex réseau IP et transport TCP)

Inconvénient: il faut lister les règles d'interdiction de façon exhaustive

Règle par défaut (politique)

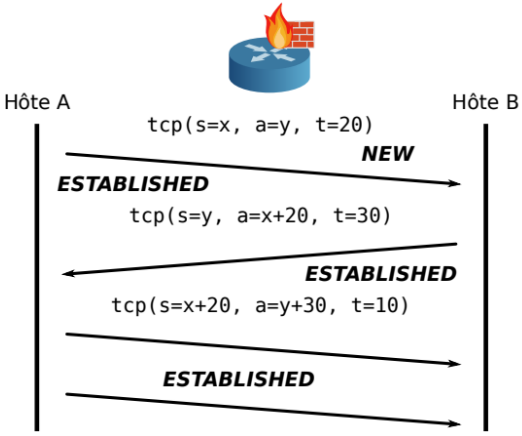
Comportement lorsque aucune règle ne s'applique

- Rejet par défaut → sécurité par défaut (banques,armée…)
- Acceptation par défaut → sûreté par défaut (systèmes critiques/temps réel)

Le + souvent: rejet par défaut (ex. box internet)

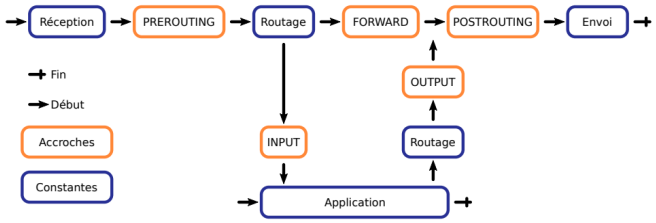
Pare-feu avec suivi d'état (*stateful*)

Suivi des interactions protocolaires niveau réseau à application pour filtrer plus finement.



Les pare-feux peuvent être matériels (+ performants) ou logiciels (très flexibles).

Netfilter



PREROUTING : réception d'un paquet sur une interface
POSTROUTING : envoi d'un paquet sur une interface
FORWARD : relayage d'un paquet réseau (mode routeur)
INPUT : le paquet en entrée est adressée localement
OUTPUT : une application envoie un paquet

Plusieurs tables iptables (exécutées dans cette ordre) :
mangle : modification de paquets
- Modification de champs à la volée et sans suivi d'état (TTL, …)
- Chaînes PREROUTING, POSTROUTING et FORWARD

nat :
- mise en place du source NAT ou destination NAT
- Chaînes PREROUTING et POSTROUTING

filter : pare-feu
- Définit les règles de filtrage : cibles DROP ou ACCEPT
- Journalise les événements : cibles LOG
- Chaînes INPUT, OUTPUT et FORWARD

Exemples commandes iptables

1	12.0.0.0/8	13.1.3.0/24	*	80	accepter
2	13.1.3.0/24	12.0.0.0/8	80	*	accepter
3	Règle par défaut				rejeter

1. `iptables -t filter -A FORWARD -s 12.0.0.0/8 -d 13.1.3.1 -p tcp --dport 80 -j ACCEPT`
2. `iptables -t filter -A FORWARD -s 13.1.3.0/24 -d 12.0.0.0 -p tcp --sport 80 -j ACCEPT`
3. `iptables -t filter -P FORWARD DROP`

Acceptation si autorisé
```
iptables -t filter -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Configuration règle de sortie
```
iptables -t filter -A FORWARD -s 13.1.3.0/24 -d 12.0.0.0 -p tcp --sport 80 -m state --state NEW -j ACCEPT
```

Configuration de la règle par défaut
```
iptables -t filter -P FORWARD DROP
```

TP pare-feu

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

iptables -t filter -A INPUT -i lo -j ACCEPT
iptables -t filter -A OUTPUT -o lo -j ACCEPT

#iptables -t filter -A FORWARD -d $DMZ -p tcp --dport 80 -j ACCEPT
#iptables -t filter -A FORWARD -s $DMZ -p tcp --sport 80 -j ACCEPT

iptables -t filter -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -t filter -A FORWARD -d $DMZ -p tcp --dport 80 -j ACCEPT -m state --state NEW
iptables -t filter -A FORWARD -d $DMZ -p tcp --dport 443 -j ACCEPT -m state --state NEW

iptables -t filter -A FORWARD -d $DMZ -p tcp --dport 25 -j ACCEPT -m state --state NEW
iptables -t filter -A FORWARD -s $DMZ -d $INTNET -p tcp --dport 25 -j ACCEPT -m state --state NEW

iptables -t filter -A FORWARD -d $DMZ -p tcp --dport 53 -j ACCEPT -m state --state NEW
iptables -t filter -A FORWARD -d $DMZ -p udp --dport 53 -j ACCEPT -m state --state NEW

iptables -t filter -A FORWARD -d $DMZ -p icmp -j ACCEPT -m state --state NEW
```

TP buffer overflow

```
.global shellcode
shellcode:
        push    $0x3b                           #59 : code de execve
        pop     %rax
        xor     %rsi, %rsi                      #deuxième argument = null
        xor     %rdx, %rdx              #troisième argument = null
        mov     $0x68732f6e69622faa, %rdi  #"/bin/sh" en hexa, renversé car little endian
        shr     $0x8, %rdi                      # on ajoute les 0
        push    %rdi
        push    %rsp
        pop     %rdi                            #premier argument, rdi stocke l'adresse vers /bin/sh
        syscall
```