Rapport de TP : MD5 Collision Attack

Nom des étudiants : Chater Ahmed et Daouda David Coulibaly
Date : 22/12/2024

Introduction

Ce rapport présente les résultats du TP sur les attaques de collision MD5. L'objectif principal était de comprendre la vulnérabilité du MD5 à des attaques de collision, où deux fichiers différents peuvent avoir le même hash MD5. Cette démonstration permet de mettre en évidence l'importance de la résistance aux collisions pour la sécurité des fonctions de hachage.

Environnement du TP

L'environnement utilisé pour ce TP était une machine virtuelle Ubuntu 20.04 configurée avec les outils nécessaires pour effectuer l'attaque de collision MD5. Le programme principal utilisé pour générer les collisions est `md5collgen`, un outil développé par Marc Stevens, installé dans le répertoire `/usr/bin`.

```
[12/22/24]seed@VM:~$ cd /usr/
[12/22/24]seed@VM:/usr$ cd /bin/
[12/22/24]seed@VM:/bin$ ls
'['                                 migrate-pubring-from-classic-gpg
 aa-enabled                         mimeopen
 aa-exec                            mimetype
 aconnect                           min12xxw
 acpi_listen                        minfo
 add-apt-repository                 mkdir
 addpart                            mkfifo
 addr2line                          mkfontdir
 alsabat                            mkfontscale
 alsaloop                           mkisofs
 alsamixer                          mkmanifest
 alsatplg                           mk_modmap
 alsaucm                            mknod
 amidi                              mkpasswd
 amixer                             mksquashfs
 amuFormat.sh                       mktemp
 apg                                mkzftree
 apgbfm                             mlabel
 aplay                              mmcli
 aplaymidi                          mmd
```

Outils utilisés

- **md5collgen** : Génération de collisions MD5
- **bless** : Éditeur hexadécimal pour visualiser et modifier les fichiers binaires
- **diff, md5sum** : Vérification des différences et des hash MD5

```
mcookie                    zgrep
mcopy                      zip
md5collgen                 zipcloak
md5sum                     zipdetails
md5sum.textutils           zipgrep
mdel                       zipinfo
mdeltree                   zipnote
mdig                       zipsplit
mdir                       zisdecode
```

Tâches réalisées

1. Génération de deux fichiers distincts ayant le même hash MD5

Dans cette tâche, nous avons utilisé l'outil `md5collgen` pour générer deux fichiers différents mais ayant le même hash MD5. Nous avons utilisé un fichier préfixe avec un contenu arbitraire et avons généré deux fichiers de sortie, `out1.bin` et `out2.bin`.

Commande exécutée :

```
bash
Copier le code
$ md5collgen -p prefix.txt -o out1.bin out2.bin
```

```
[12/22/24]seed@VM:~/Documents$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 929ae445e2f75bcfd93085e3789623d3

Generating first block: ..........
Generating second block: S00.........................
Running time: 13.4644 s
```

```
[12/22/24]seed@VM:~/Documents$ ls -l out1.bin out2.bin
-rw-rw-r-- 1 seed seed 192 Dec 22 07:07 out1.bin
-rw-rw-r-- 1 seed seed 192 Dec 22 07:07 out2.bin
[12/22/24]seed@VM:~/Documents$
```

Nous avons ensuite vérifié les différences entre les fichiers générés avec la commande `diff` et avons confirmé que leurs hash MD5 étaient identiques à l'aide de `md5sum`.

Commandes vérification :

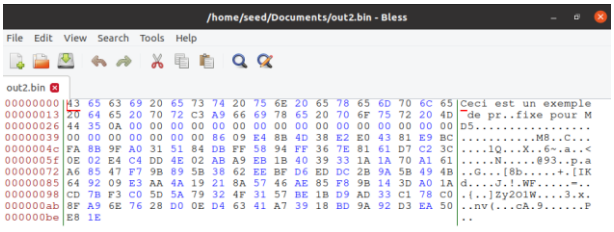
```
bash
Copier le code
$ diff out1.bin out2.bin
```

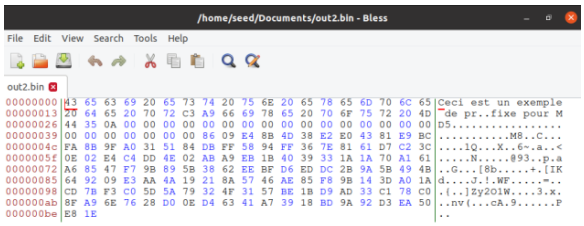
```
[12/22/24]seed@VM:~/Documents$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[12/22/24]seed@VM:~/Documents$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
```

```
$ md5sum out1.bin
$ md5sum out2.bin
```

```
[12/22/24]seed@VM:~/Documents$ md5sum out1.bin
e3cc6de65b8aa3558fd5a3f5311f1aa9  out1.bin
[12/22/24]seed@VM:~/Documents$ md5sum out2.bin
e3cc6de65b8aa3558fd5a3f5311f1aa9  out2.bin
```

Observation





- Les deux fichiers sont binaires, et leur contenu est essentiellement différent, mais leur hash MD5 est identique.
- Nous avons utilisé un éditeur hexadécimal pour explorer ces fichiers et avons observé les sections de données modifiées pour générer la collision.

2. Compréhension de la propriété de MD5

```
[12/22/24]seed@VM:~/Documents$ echo "Ceci est un suffixe pour les tests MD5" > suffix.txt
```

Nous avons exploré la propriété de MD5 qui stipule que si deux messages M et N ont le même hash, alors pour tout suffixe T, MD5(M || T) = MD5(N || T). Cette propriété a été vérifiée par l'expérimentation suivante :

Commande exécutée :

```
[12/22/24]seed@VM:~/Documents$ cat out1.bin suffix.txt > file1.bin
[12/22/24]seed@VM:~/Documents$ cat out2.bin suffix.txt > file2.bin
```

```
bash
Copier le code
$ cat file1 file2 > file3
```

Observation

```
[12/22/24]seed@VM:~/Documents$ md5sum file1.bin
15fe419544a943b85f3a34b50f92b524  file1.bin
[12/22/24]seed@VM:~/Documents$ md5sum file2.bin
15fe419544a943b85f3a34b50f92b524  file2.bin
[12/22/24]seed@VM:~/Documents$
```

Nous avons démontré que l'ajout du même suffixe à deux fichiers ayant le même hash MD5 entraîne le même hash après concaténation.

3. Génération de deux exécutables différents avec le même hash MD5

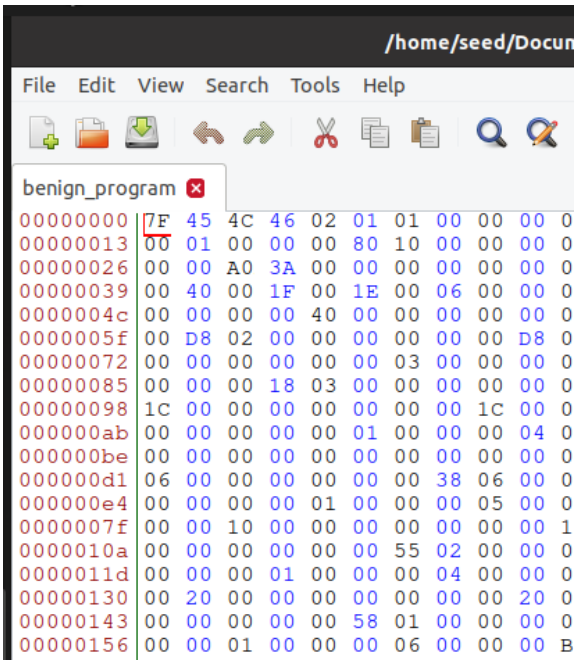
```
[12/22/24]seed@VM:~/Documents$ nano program.c
```

Nous avons modifié le contenu du tableau `xyz` dans un programme C donné et avons compilé deux versions du programme avec des valeurs différentes dans le tableau. Nous avons ensuite utilisé l'outil `md5collgen` pour générer une collision MD5.

```
 GNU nano 4.8                  program.c
#include <stdio.h>

// Tableau à remplir avec des données arbitraires
unsigned char xyz[200] = {
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, // 'A' en ASCII
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    // Ajoutez plus de données si nécessaire pour remplir les 200 >
};

int main() {
    for (int i = 0; i < 200; i++) {
        printf("%x ", xyz[i]);
    }
    printf("\n");
    return 0;
}
```

Commande exécutée :

```
[12/22/24]seed@VM:~/Documents$ md5collgen -p prefix -o block1.bin block2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'block1.bin' and 'block2.bin'
Using prefixfile: 'prefix'
Using initial value: bbb030508cdc61835849dd4ae153a192

Generating first block: .....
Generating second block: S10.......
Running time: 5.24682 s
```




```
[12/22/24]seed@VM:~/Documents$ head -c 100 benign_program > prefix
[12/22/24]seed@VM:~/Documents$ tail -c 100 benign_program > suffix

[12/22/24]seed@VM:~/Documents$ cat prefix block1.bin suffix
[12/22/24]seed@VM:~/Documents$ cat prefix block2.bin suffix
```

```
bash
Copier le code
$ md5collgen -p prefix.c -o out1.out out2.out
```

Observation

```
[12/22/24]seed@VM:~/Documents$ md5sum benign_program_1
f5284a423694622ca32475c8f001ebc4  benign_program_1
[12/22/24]seed@VM:~/Documents$ md5sum benign_program_2
f5284a423694622ca32475c8f001ebc4  benign_program_2
[12/22/24]seed@VM:~/Documents$
```

Les deux exécutables partagent le même hash MD5, bien qu'ils comportent des différences dans le tableau `xyz`.

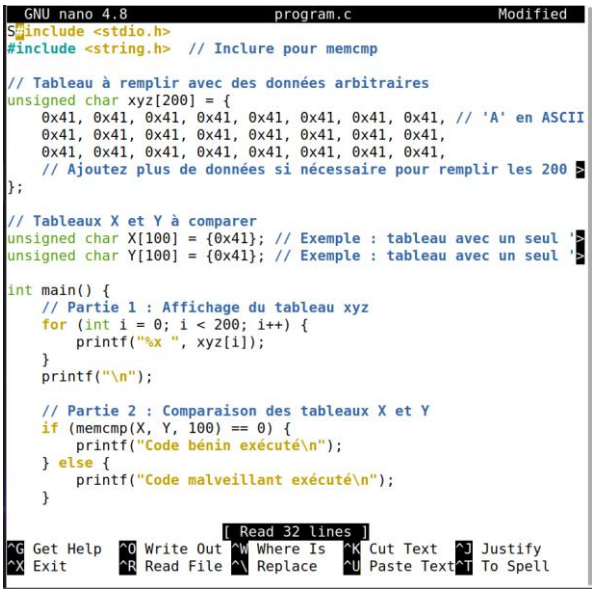
4. Création de deux programmes exécutables avec des comportements différents mais le même hash MD5

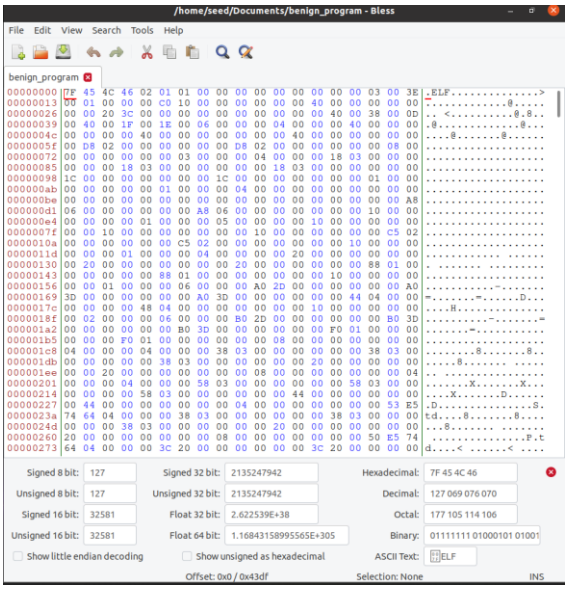
Dans cette tâche, nous avons développé deux versions d'un programme C : une version bénigne et une version malveillante. Les deux programmes partagent un comportement similaire, mais l'exécution de l'une des versions provoque un effet malveillant.

Nous avons utilisé la technique de collision MD5 pour générer deux versions du programme avec le même hash, tout en changeant la logique interne à l'aide de tableaux X et Y.

Code source utilisé :

```
c
Copier le code
unsigned char xyz[200] = { /* valeurs arbitraires */ };

int main() {
    if (memcmp(X, Y, sizeof(X)) == 0) {
        // code bénin
    } else {
        // code malveillant
    }
}
```






```
[12/22/24]seed@VM:~/Documents$ head -c 100 benign_program > prefix
[12/22/24]seed@VM:~/Documents$ tail -c 100 benign_program > suffix

[12/22/24]seed@VM:~/Documents$ md5collgen -p prefix -o block1.bin block2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'block1.bin' and 'block2.bin'
Using prefixfile: 'prefix'
Using initial value: 36ab84fb0fa5fa72080e9af2aaf51e7e

Generating first block: ............................................................
Generating second block: W...........
Running time: 115.943 s
[12/22/24]seed@VM:~/Documents$
[12/22/24]seed@VM:~/Documents$ cat prefix block1.bin suffix > benign_program_1
[12/22/24]seed@VM:~/Documents$ cat prefix block2.bin suffix > benign_program_2
```

Observation

```
[12/22/24]seed@VM:~/Documents$ md5sum benign_program_1
069bb22820c70b4b0bbdc22cd97f0af0  benign_program_1
[12/22/24]seed@VM:~/Documents$ md5sum benign_program_2
5c063c0fa50a9535fa492bfe585c3e46  benign_program_2
[12/22/24]seed@VM:~/Documents$
```

Les deux versions du programme ont le même hash MD5 mais exécutent des comportements différents en fonction des valeurs dans les tableaux X et Y.

Conclusion

Ce TP a permis de démontrer de manière pratique la vulnérabilité du MD5 à des attaques de collision. Nous avons généré deux fichiers, puis deux exécutables, ayant le même hash MD5 mais exécutant des comportements différents, illustrant le potentiel de ces attaques dans des contextes de certification et de vérification d'intégrité.