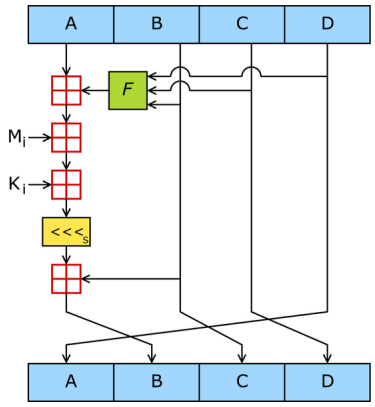
MD5 collision 实验报告

57119101 王晨阳

2021年7月31日

实验原理



$$\begin{cases} F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z) \\ H(X,Y,Z) = X \oplus Y \oplus Z \\ I(X,Y,Z) = Y \oplus (X \vee \neg Z) \end{cases}$$

四个函数每个作用 16 轮，得到最后的结果。

Task 1: Generating Two Different Files with the Same MD5 Hash

首先创建 `prefix.txt`

```
touch prefix.txt
vim prefix.txt
```

例如，我们修改内容为

```
hail hydra
```

然后生成两个 md5 相同的文件

```
md5collgen -p prefix.txt -o out1.bin out2.bin
```

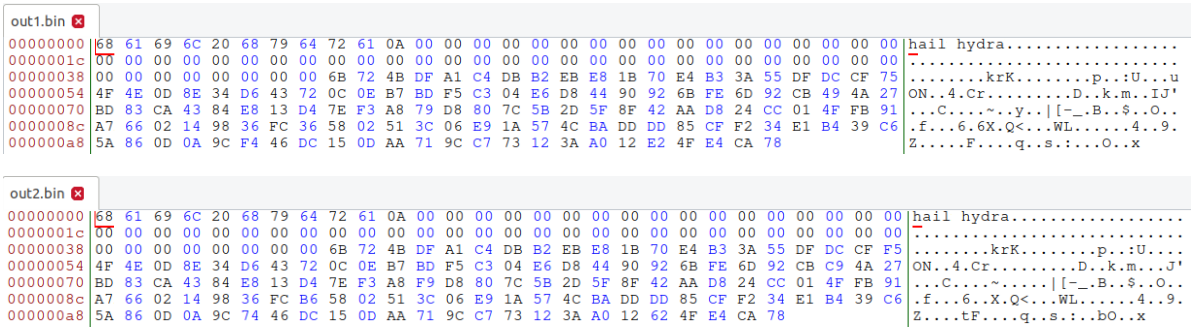
验证我们的文件是否相同、md5是否相同

```
diff out1.bin out2.bin
md5sum out1.bin
md5sum out2.bin
```

得到如下结果

```
[07/30/21]seed@VM:~/.../md5$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[07/30/21]seed@VM:~/.../md5$ md5sum out1.bin
9ff3d393385cc4b1074a331a1bb499d2  out1.bin
[07/30/21]seed@VM:~/.../md5$ md5sum out2.bin
9ff3d393385cc4b1074a331a1bb499d2  out2.bin
```

我们分别查看 out1.bin 和 out2.bin



> Question 1 If the length of your prefix file is not multiple of 64, what is going to happen?

补零至 64 Byte。

> Question 2 Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.

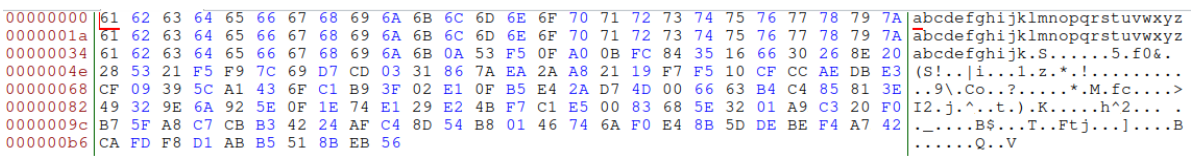
将 prefix.txt 改为

```
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijk
```

这里一共 63 个字母，加上文件结束符 `0A` 正好 64 Byte。

```
md5collgen -p prefix.txt -o out1.bin out2.bin
```

生成文件后如下所示



可以看到没有补零了。

> Question 3 Are the data (128 bytes) generated by md5collgen completely different for the two
> output files? Please identify all the bytes that are different.

例如第一个例子中，有 3 个 Byte 不同。经过多次尝试发现，这些不同的数量和位置不固定。

Task 2: Understanding MD5’s Property

我们对刚刚的两个 md5 相同的文件分别加上一个后缀，然后查看它们的 md5

```
echo hello >> out1.bin
echo hello >> out2.bin
md5sum out1.bin out2.bin
```

```
[07/30/21]seed@VM:~/.../md5$ echo hello >> out1.bin
[07/30/21]seed@VM:~/.../md5$ echo hello >> out2.bin
[07/30/21]seed@VM:~/.../md5$ md5sum out1.bin out2.bin
868ce88de5b9855d09c21c449909e105  out1.bin
868ce88de5b9855d09c21c449909e105  out2.bin
```

可以看到，md5 相同的文件加上相同后缀后，md5 依然相同。

Task 3: Generating Two Executable Files with the Same MD5 Hash

新建 pro.c

```
touch pro.c
```

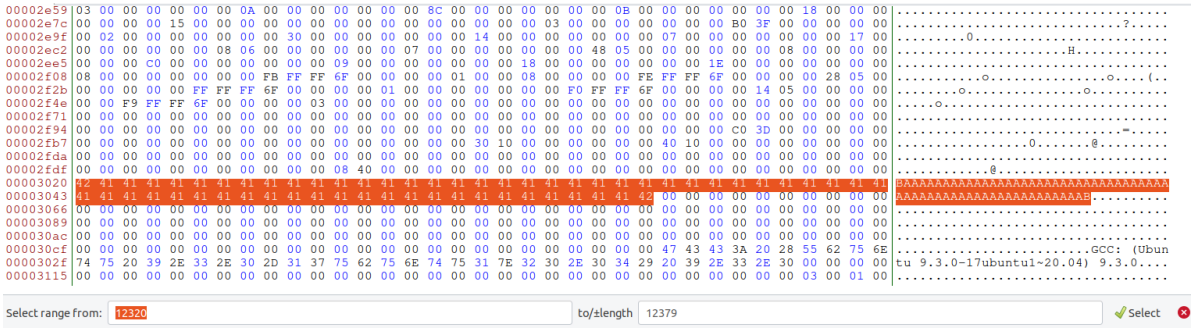
修改内容如下

```
#include <stdio.h>
unsigned char xyz[200] = {
    'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B'
};
int main()
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

编译

```
gcc -o pro pro.c
```

定位到刚刚的字符串存储在 `0x3020` 位置



我们不妨截取到 12340 位置

```
head -c 12340 pro > prefix
```

计算得到在 12320 到 12379 范围内，12352 为 64 的倍数，因此我们把12352 后面的截取出来

```
tail -c +12353 pro > suffix
```

然后对 prefix 生成 md5 相同的两个文件

```
md5collgen -p prefix -o prefix1 prefix2
```

把刚刚的尾巴接到这两个文件后面

```
cat suffix >> prefix1
cat suffix >> prefix2
```

赋予执行权限

```
chmod +x prefix1
chmod +x prefix1
```

运行

```
[07/30/21]seed@VM:~/.../md5$ ./prefix1
4241414141414141414141414141414141414141000000000007bcbaddb68638c84258c8ef6a826
ba28bb3043f4c3223389537e82eed39493d024cca58cd448c98cb4d45f2f8317e397a6932ddbeb95
a39f88f4a1f1d213f644f65e270d9af707356328d83a23a3d0cc6e222d25fb74bc0474fc6fc164ec
c02d94b08afea8c3eff85cb688be94c1711c8f50592925e9fd76e128bb414141414141414141414
4141414141414141414141414141414141420000000000000
[07/30/21]seed@VM:~/.../md5$ ./prefix2
4241414141414141414141414141414141414141000000000007bcbaddb68638c84258c8ef6a826
ba28bb304374c3223389537e82eed39493d024cca58cd448c98cb4d45f2f8b17e397a6932ddbeb95
a39f88f421f1d213f644f65e270d9af707356328d83a23a3d0ccee222d25fb74bc0474fc6fc164ec
c02d94b08afea8c3eff85c3688be94c1711c8f50592925e97d76e128bb414141414141414141414
4141414141414141414141414141414141420000000000000
```

可以看到，两个输出是不同的

```
./prefix1 > prefix1.out
./prefix2 > prefix2.out
diff -q prefix1.out prefix2.out
```

```
[07/30/21]seed@VM:~/.../md5$ ./prefix1 > prefix1.out
[07/30/21]seed@VM:~/.../md5$ ./prefix2 > prefix2.out
[07/30/21]seed@VM:~/.../md5$ diff -q prefix1.out prefix2.out
Files prefix1.out and prefix2.out differ
```

Task 4: Making the Two Programs Behave Differently

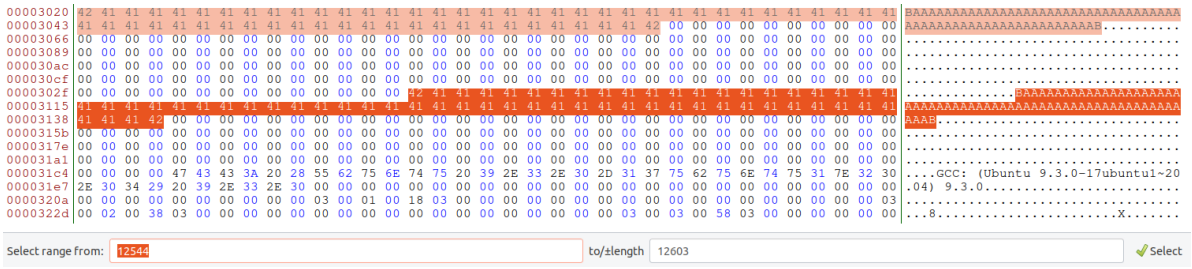
构造 `origin.c` 如下

```c
#include <stdio.h>
unsigned char a[200] = {
    'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B'
};
unsigned char b[200] = {
    'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B'
};
int main()
{
    int i;
    int isSame=1;
    for(i = 0; i < 200; i++)
    {
        if(a[i]!=b[i])
            isSame=0;
    }
    if(isSame)
        printf("run benign code\n");
    else
        printf("run malicious code\n");
}
```

编译

```
gcc -o origin origin.c
```

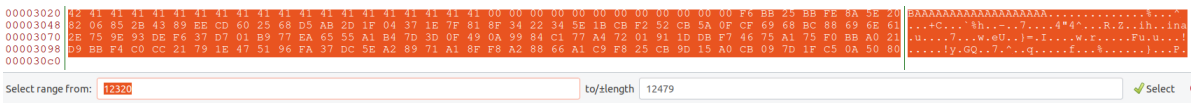
找到两个数组位置



与上一个 task 同样的方法，我们构造两个 md5 相同的文件

```
head -c 12340 origin > prefix
md5collgen -p prefix -o prefix1 prefix2
```

查看 prefix1



截取

```
tail -c +12320 prefix1 > middle # 截取生成的字符串
tail -c +12768 origin > suffix # 截取第二个字符串（不含）后面的内容
head -c 12543 origin > tmp1 # 截取第二个字符串（不含）前面的内容
tail -c 63 tmp1 > tmp # 截取到需要填充的 0x00
cat tmp >> prefix1
cat tmp >> prefix2
cat middle >> prefix1
cat middle >> prefix2
cat tmp >> prefix1
cat tmp >> prefix2
cat suffix >> prefix1
cat suffix >> prefix2
chmod +x prefix1
chmod +x prefix2
```

分别运行并检查 md5

```
[07/30/21]seed@VM:~/.../md5$ ./prefix1
run benign code
[07/30/21]seed@VM:~/.../md5$ ./prefix2
run malicious code
[07/30/21]seed@VM:~/.../md5$ md5sum prefix1 prefix2
82d3b9f2e3110eeb62d5a029acef283c  prefix1
82d3b9f2e3110eeb62d5a029acef283c  prefix2
```

可以看到，它们运行了不同的代码，但 md5 是相同的。

实验总结

前面 3 个 Task 非常简单。最后一个接来接去的不搞乱了、把该截取多少个想清楚了，就做出来了。