Cryptography:
MD5 Collision Attack Lab

Batyi Amatov (amatobat@b-tu.de)

Table of Content

Abstract:

We know that it is very difficult to generate same hash function value from different content. But in this lab our main goal is to study and understand the impact of collision attacks. We have created two different files/programs which are not identical to each other. However from the different categories of files/program we are trying to generate collision attacks that share the same MD5 hash value but have completely different behaviors.

1. The Tasks Description:

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

Our task is to generate two different files with the same MD5 hash values. For this purpose we have used the md5collgen program to generate two files with the same hash value where the same prefix has been shared.

In figure 1, we have our initial input file with share prefix.

Figure 1: A prefix file with arbitrary content

Based on the given prefix (prefix.txt), we are generating two different files (out1.bin, out2.bin), as shown in Figure 3 and 4, which will give the same MD5 hash.

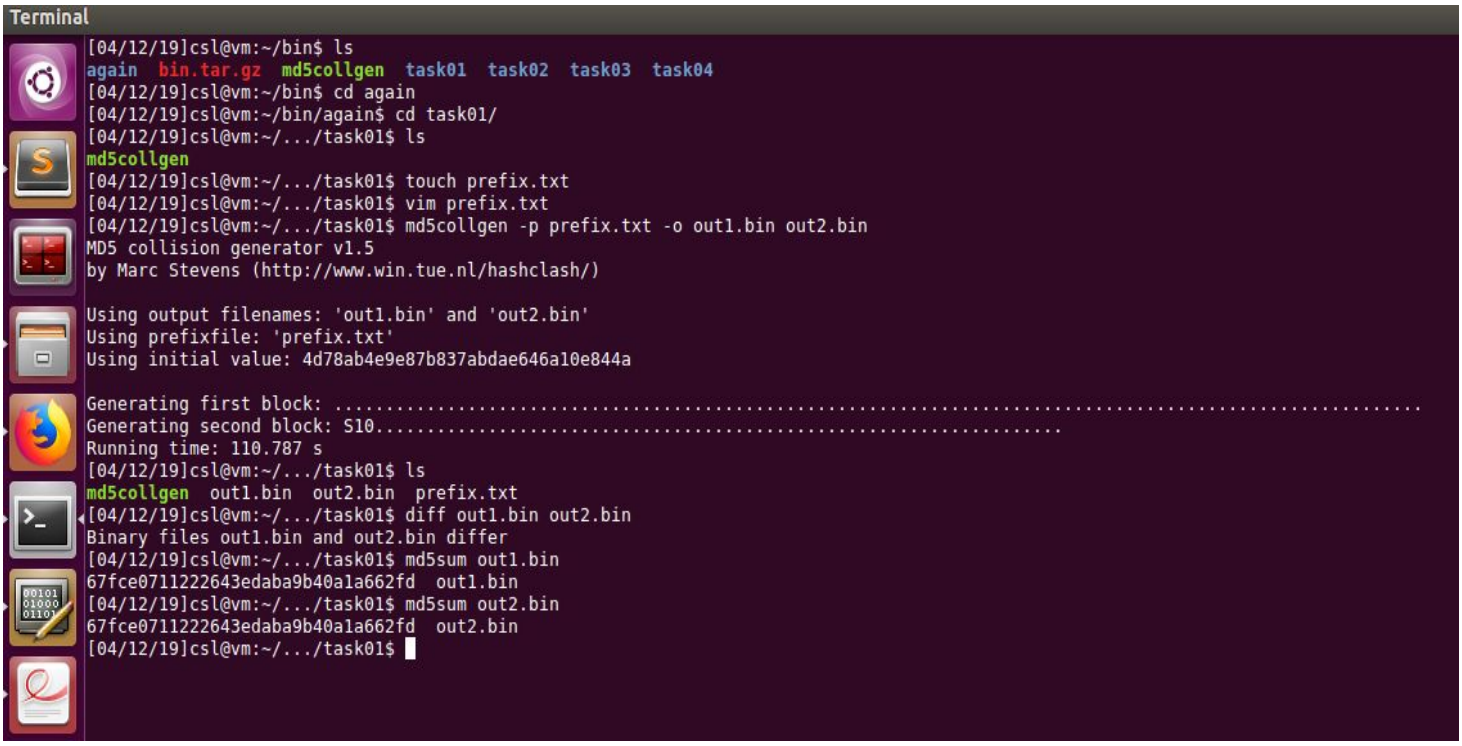

Figure 2: Command Line Instructions

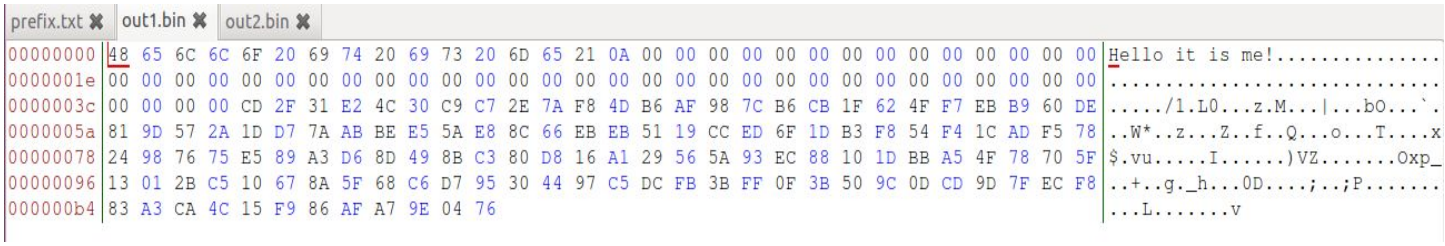

Figure 3: First output file - out1.bin

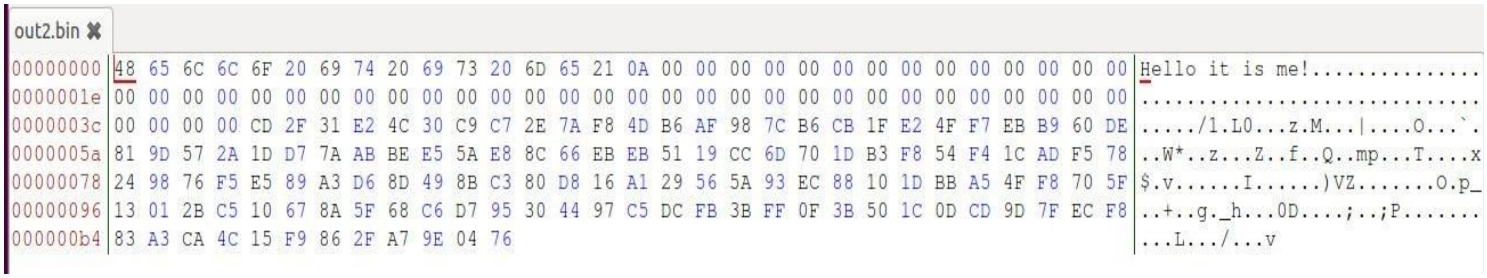

Figure 4: Second output file – out2.bin

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

In the previous case we showed that MD5 collision tool pads with zeros in order to make length of prefix files multiple of 64.

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

In the following figures 5 and 6, we observed that if the prefix file with exactly 64 bytes and MD5 collision tool didn't pad any zeros.

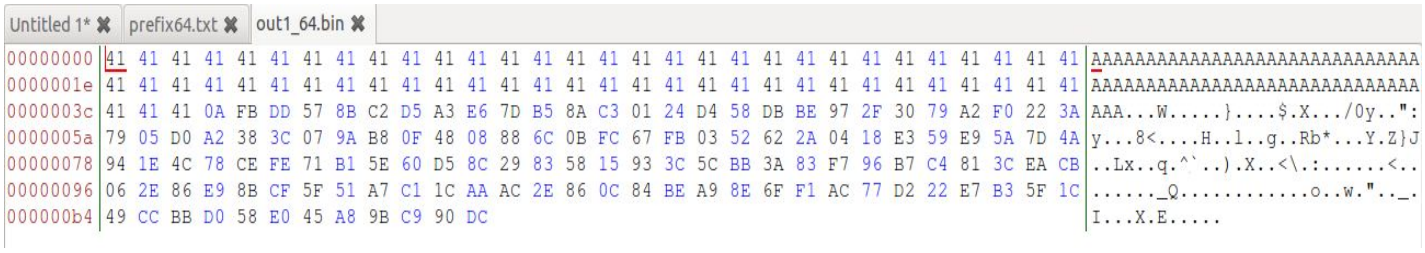

Figure 5: First output file – out1_64.bin

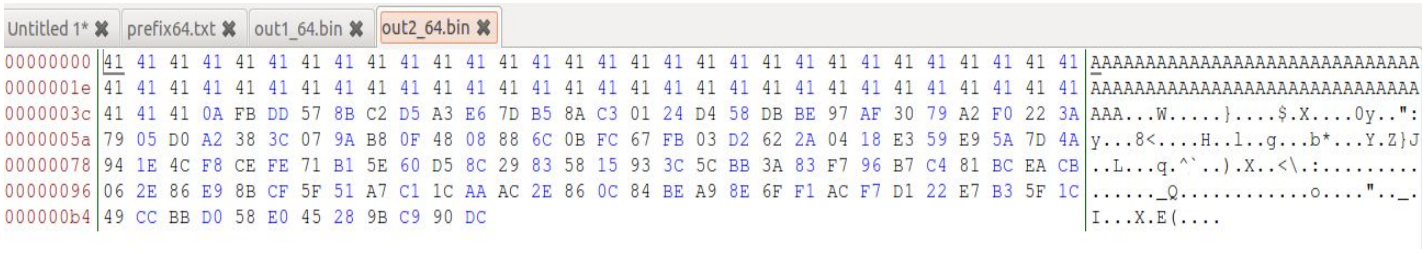

Figure 6: Second output file – out2_64.bin

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

In figures 5 and 6, we observed a slight (i.e. 32 bits) change in the two output files. We have identify the following marked bytes were changed in the two output files in figure 7 and 8.

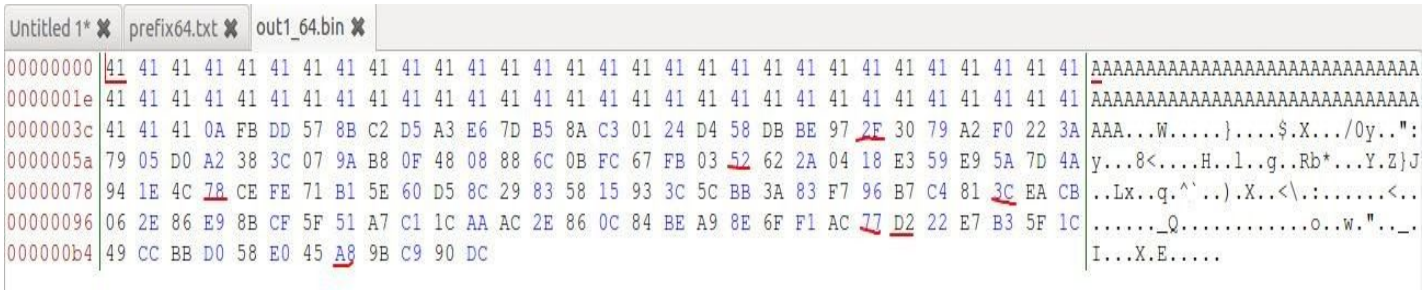

Figure 7: First output file – out1_64.bin

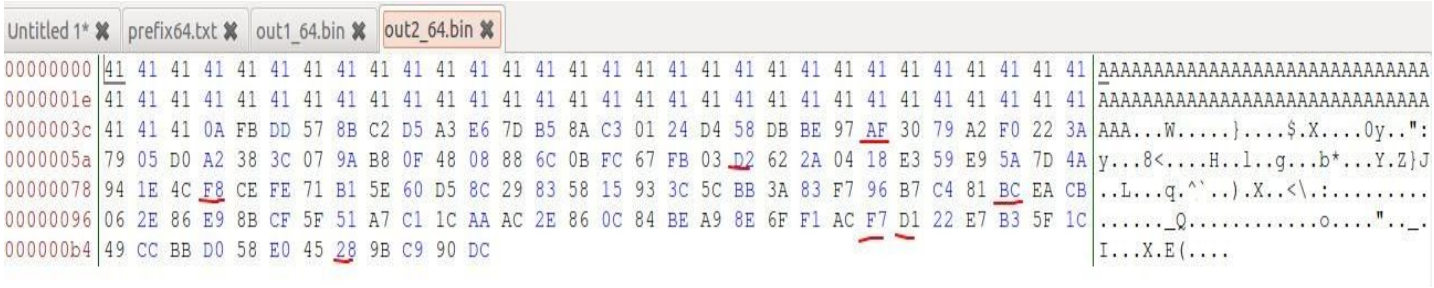

Figure 8: Second output file – out2_64.bin

Task 2: Understanding MD5's Property

Our task to identify the properties of the MD5 algorithm.

In figure 9, we observed the property of the MD5 algorithm. If adding any input by concatenation to the existing two different files which share the same MD5 hash, we would be able to get a result in two outputs that have the same hash value.

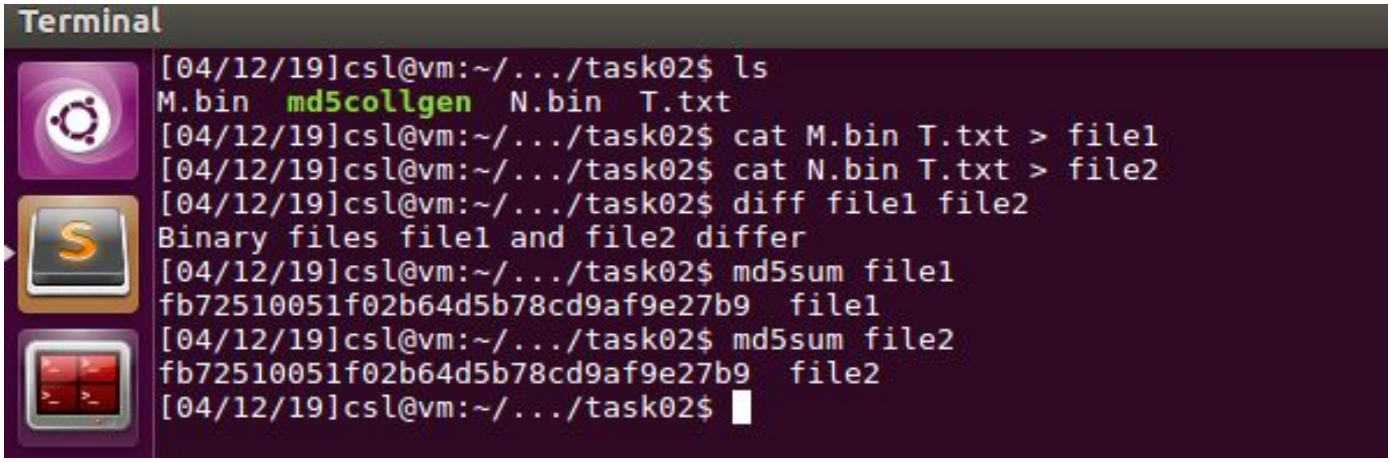

Figure 9: To understand the properties of the MD5 algorithm

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

Our task is to generate two executable files with the same MD5 Hash based on the C code provided in "**Annexure 1**". After compilation we got executable file which we need to divide into two parts: prefix which is multiple of 64 byte (which ends in the memory of array), then we left 128 byte untouched and get the rest as suffix which are shown in Figure 10, 11 and 12. For this purpose we used the following commands:

$ head -c 4160 codeExecutable > prefix
$ tail –c 3348 codeExecutable > suffix

After generating by mdcollgen on prefix we get P and Q files which share the same MD5 hash, but have differences in the last 128 bytes.

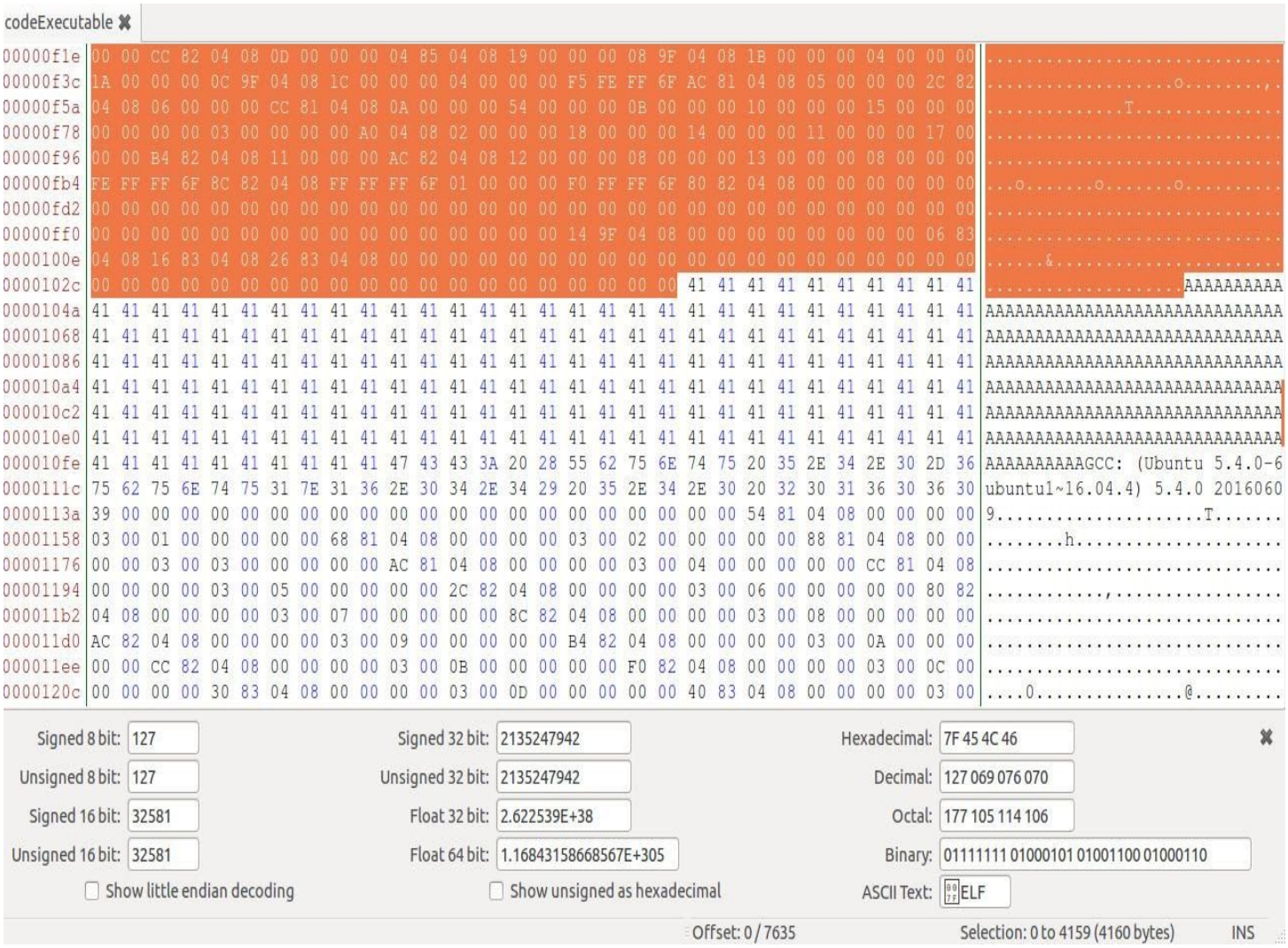

Figure 10: Executable file with prefix

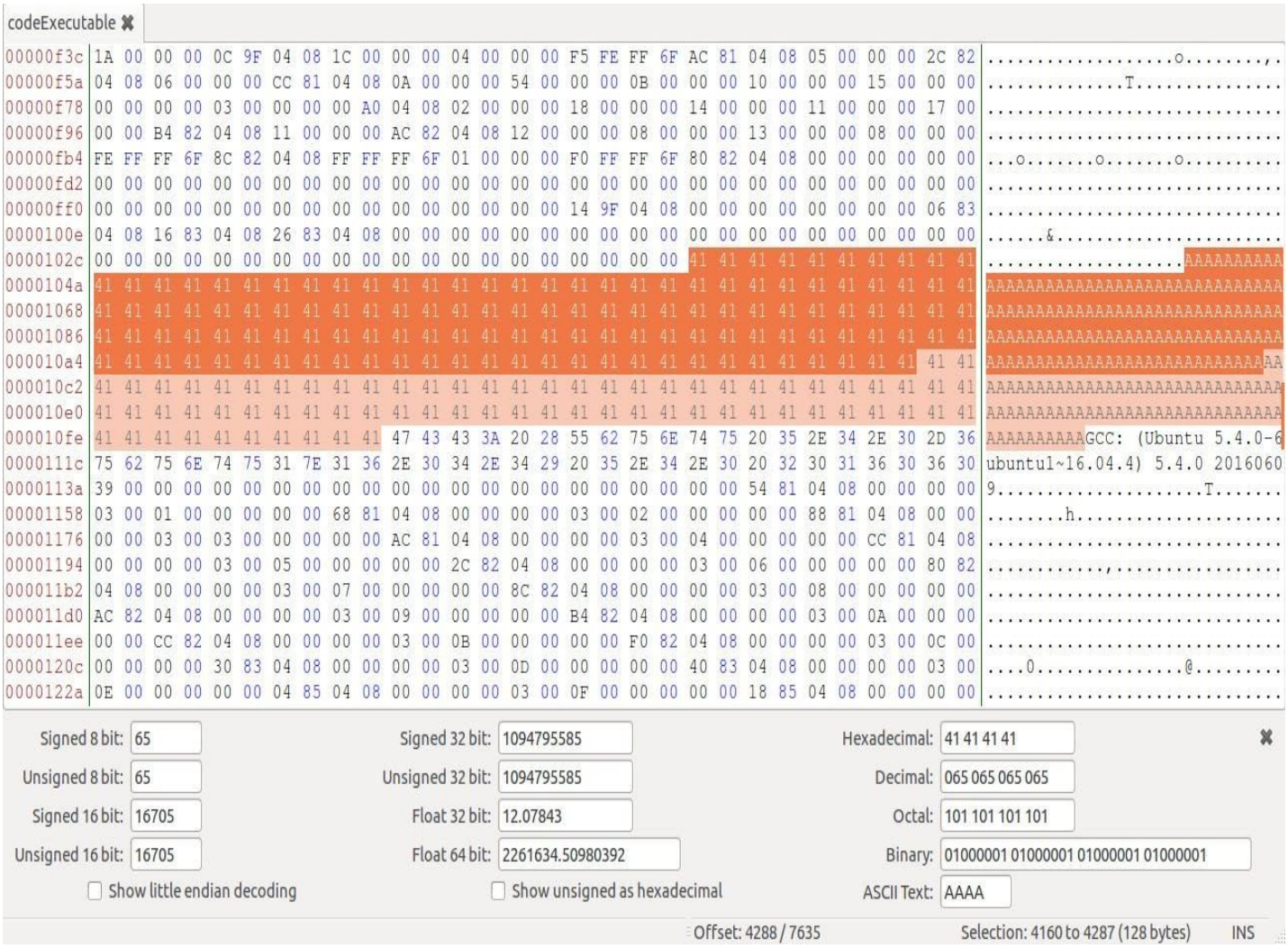

Figure 11: Executable file with 128-byte region

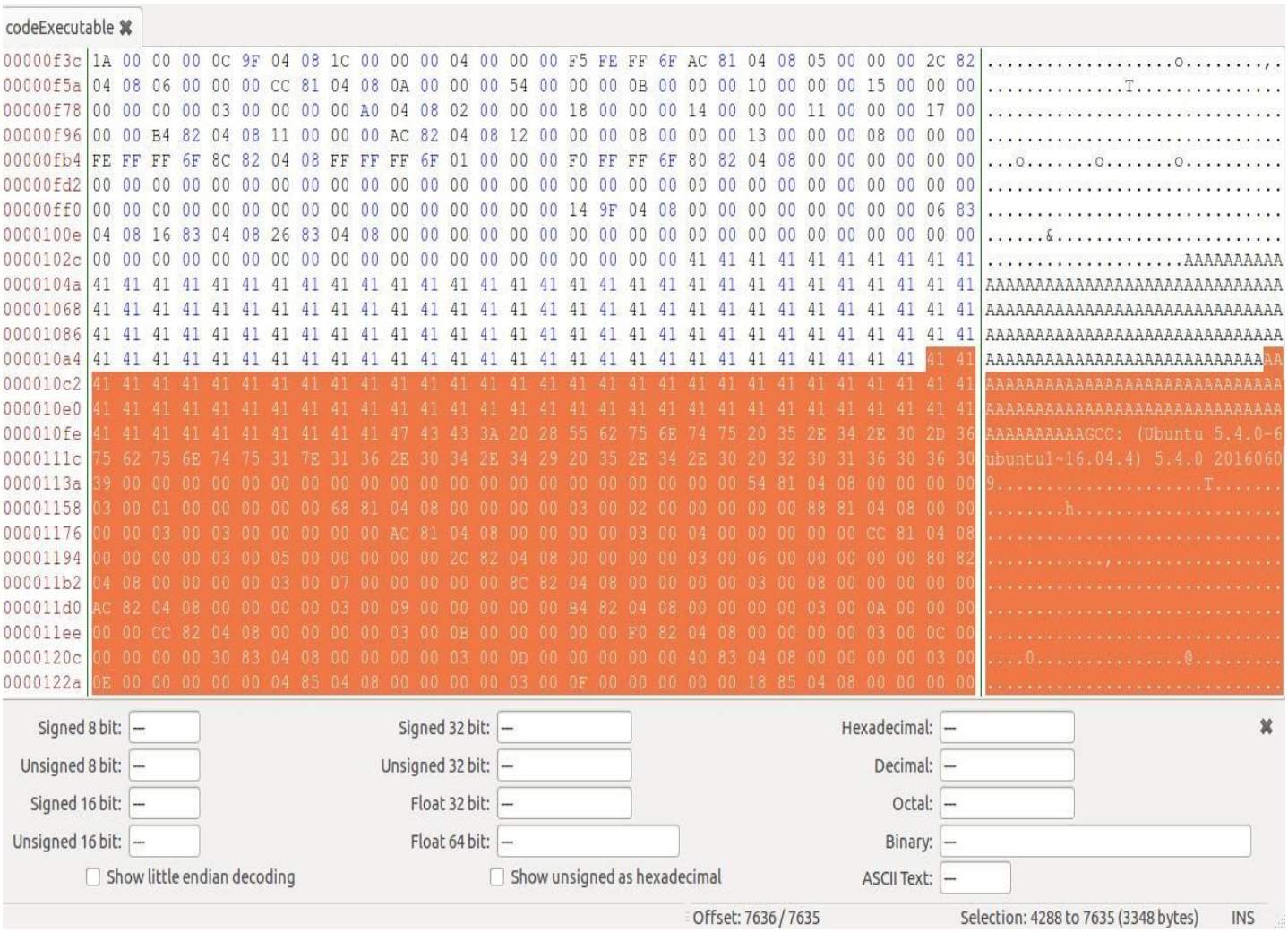

Figure 12: Executable file with suffix

After it, we are adding suffix to both files in order to gather the program again and getting files: version1 and version2, as per using command in Figure 13. After checking by md5sum they share the same MD5 hash. So, we generate two executable files with the same MD5 Hash.

```
[04/12/19]csl@vm:~/.../task03$ ls
code.c  codeExecutable  md5collgen  P.bin  prefix  Q.bin  suffix  version1
[04/12/19]csl@vm:~/.../task03$ cat P.bin suffix > version1
[04/12/19]csl@vm:~/.../task03$ cat Q.bin suffix > version2
[04/12/19]csl@vm:~/.../task03$ md5sum version1
aa45a6859179f28ba41aac36f4be02dc  version1
[04/12/19]csl@vm:~/.../task03$ md5sum version2
aa45a6859179f28ba41aac36f4be02dc  version2
[04/12/19]csl@vm:~/.../task03$
```

Figure 13: To generate two executable files with the same MD5 Hash

Task 4: Making the Two Programs Behave Differently

Our task is to prepare two different programs. One program will work properly and will go for certification, while the other program will be malicious, but will have the same MD5 hash as the first normal program. The source C code is provided in "**Annexure 2**". After compilation we got executable file from which we need to get the prefix, which is multiple of 64 byte (which ends in the memory of array), then we left 128 byte untouched and got the rest as suffix. . For this purpose we used the following commands:

$ head -c 4160 codeExecutable > prefix
$ tail –c 3548 codeExecutable > suffix

After compilation the prefix by mdcollgen, we got two files: "real" and "fake", which share the same MD5 hash, but had differences in the last 128 bytes, as shown in Figure 14.

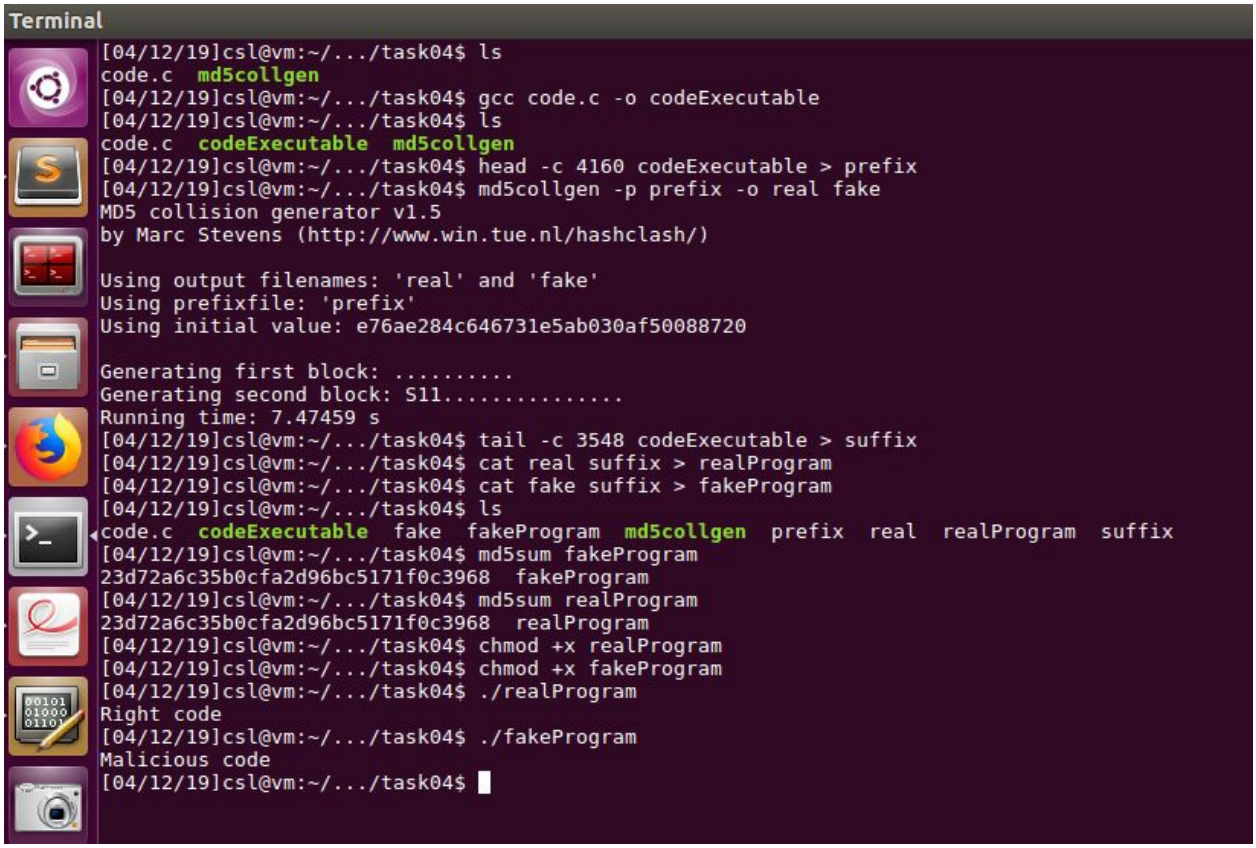

Figure 14: Process of generating two different programs with the same MD5 Hash

In Figure 15 and 16 describe that, from the "real" program we took the generated by mdcollgen tool - 128 bytes at the end and copied it into the first 128 bytes of the second array in suffix file.

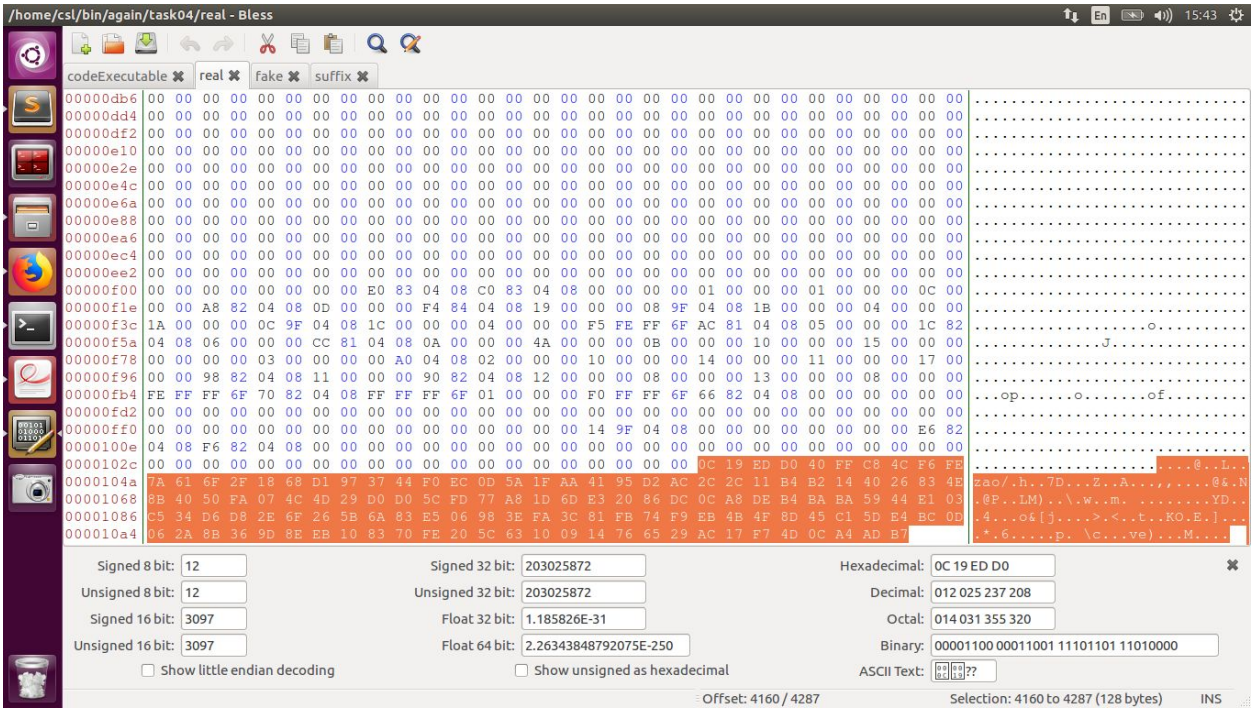

Figure 15: Copying the last 128 bytes from the real program

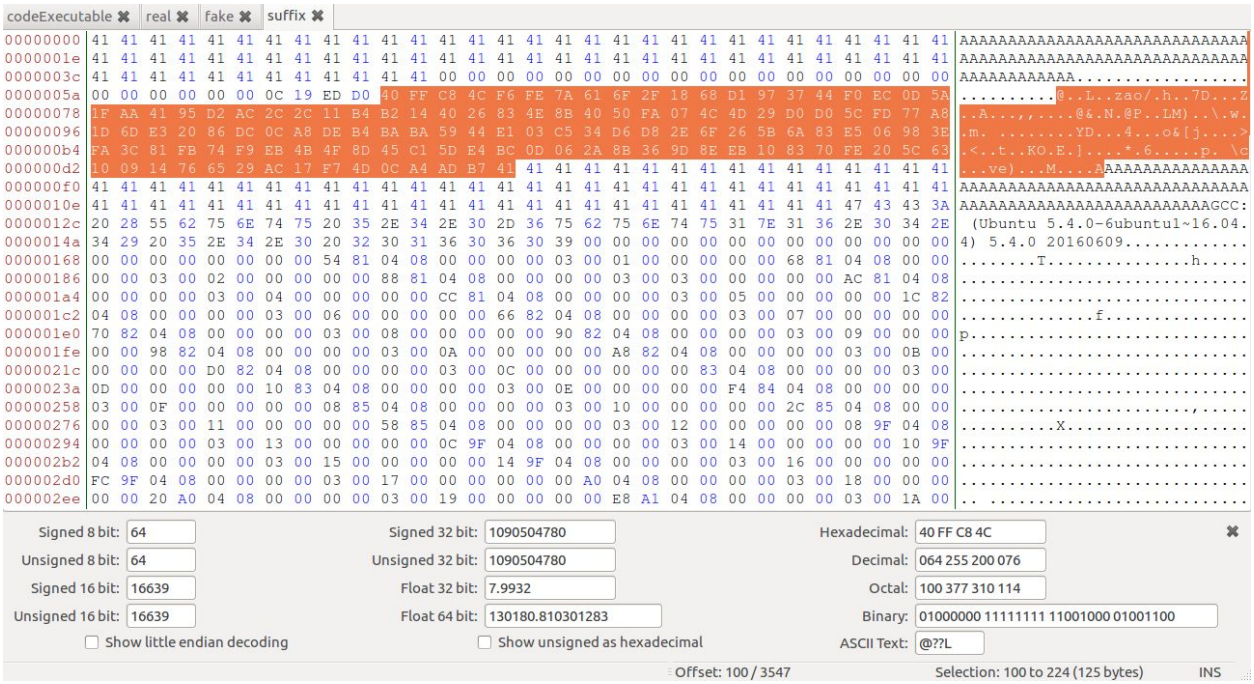

Figure 16: Putting the last 128 bytes from the real program into the first 128 bytes of the second array in suffix file.

After it, we are concatenating the updated suffix to both files (real, fake) in order to gather the program again and getting files: realProgram and fakeProgram. After checking by md5sum we have found that they shared the same MD5

hash, as shown in Figure 14. So, we generated two different behaviors of programs, based on the same source code, which share the same MD5 Hash.

2. Conclusion

Now a days the MD5 Hash function is a vulnerable hash function. The Lab is provide us the real life scenario based explanation, how the MD5 collision was happened, generating the same hash values for collision with different content and examined the properties of the hash function. Last but not the least, in the attacker point of view, it can also possible for us to generate a fake certificate with the help of real certificate whereas the content of the program were not the same that is one program is behaved and gave us correct output but the attacker program also can verify with the same certificate value due to generate same hash value (i.e collision occurs) though it will behave maliciously. So our observation is not to use the MD5 Hash function in any real world transactions.

Annexure 1

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

```
#include <stdio.h>

unsigned char xyz[200] =
{'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','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','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','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'};

int main() {

	int i;

	for(i = 0; i < 200; i++) {

		printf("%x", xyz[i]); // Print the Array contents

	}

	printf("\n");

}
```

Annexure 2

Task 4: Making the Two Programs Behave Differently

```
#include <stdio.h>

unsigned char x[200] =
{'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','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','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','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'};

unsigned char y[200] =
{'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','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','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','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'};

int main() {
    int f = 1;
    for(int i = 0; i < 200; i++) {
        if(x[i] != y[i])  // checking arrays for the equivalence
            f = 0;
    }
    if(f == 1) // Array x’s contents and Array y’s contents are the same
        printf("Right code\n");
    else  // For execute the Attacker’s code
        printf("Malicious code\n");
}
```

Reference

[1] J. Black, M. Cochran, and T. Highland. “A Study of the MD5 Attacks: Insights and Improvements”. In: Proceedings of the 13th International Conference on Fast Software Encryption. FSE’06. Graz, Austria: Springer-Verlag, 2006, pp. 262–277. ISBN: 3-540-36597-4, 978-3540-36597-6. DOI: 10.1007/11799313_17. URL: http://dx.doi.org/10.1007/11799313_17 (cit. on p. 1).

[2] M. Stevens. “On collisions for md5”. MA thesis. Eindhoven University of Technology, 2007 (cit. on p. 6).

[3] M. Stevens, E. Bursztein, et al. The first collision for full SHA-1. Cryptology ePrint Archive, Report 2017/190. https://eprint.iacr.org/2017/190. 2017.