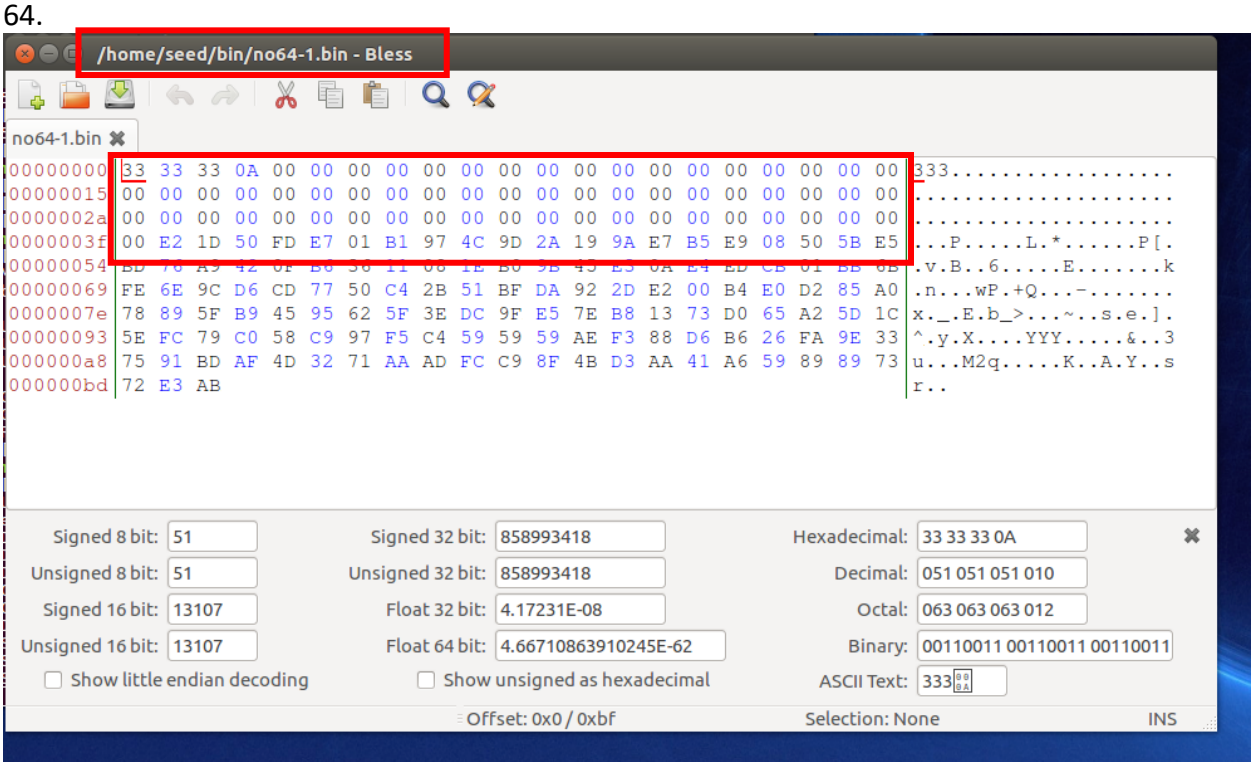
Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

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

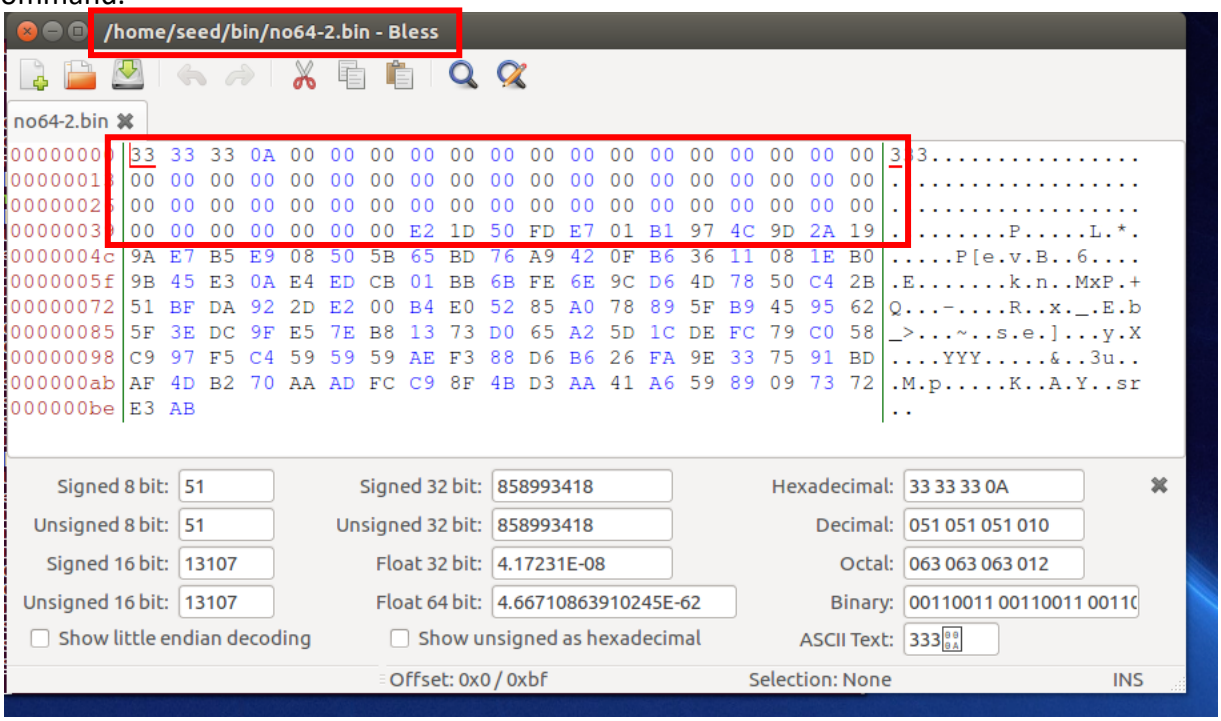
In the case where a prefix file was not a multiple of 64, we can see that the remaining byte space up to the next multiple of 64 is going to be filled with padding in the form of the character “00”. We see this occur for both files which result from running the md5collgen command.

**Screenshot 1:** Usage of padding in the form of “00” due to the prefix file not being a multiple of 64.



Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

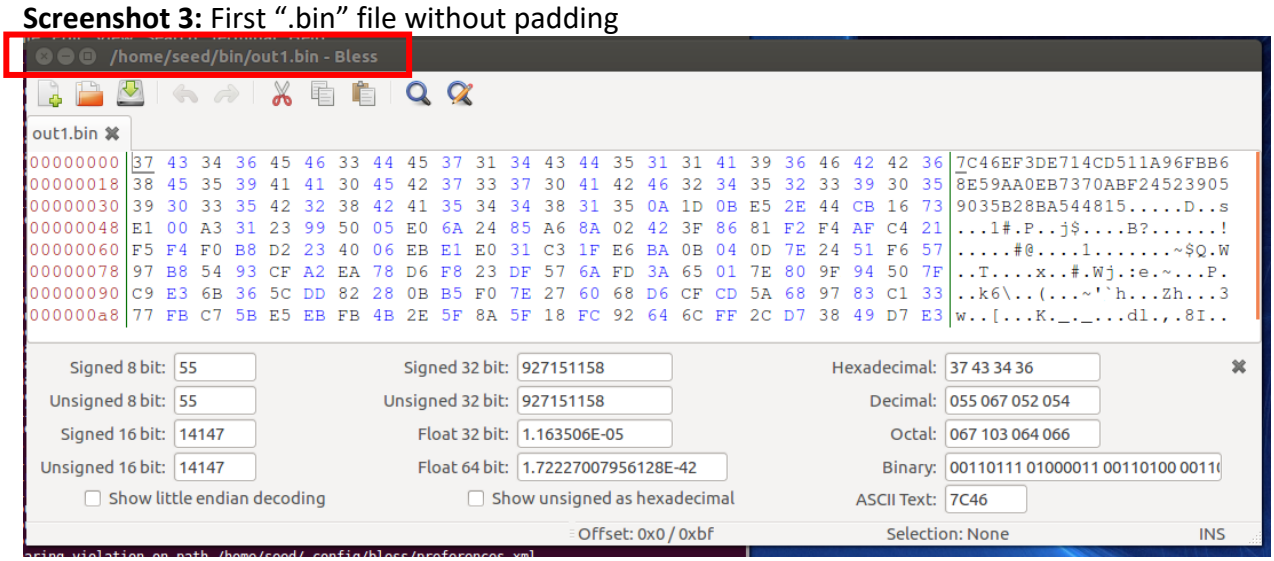
**Screenshot 2:** Usage of padding on the second ".bin" file produced by the md5collgen command.




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

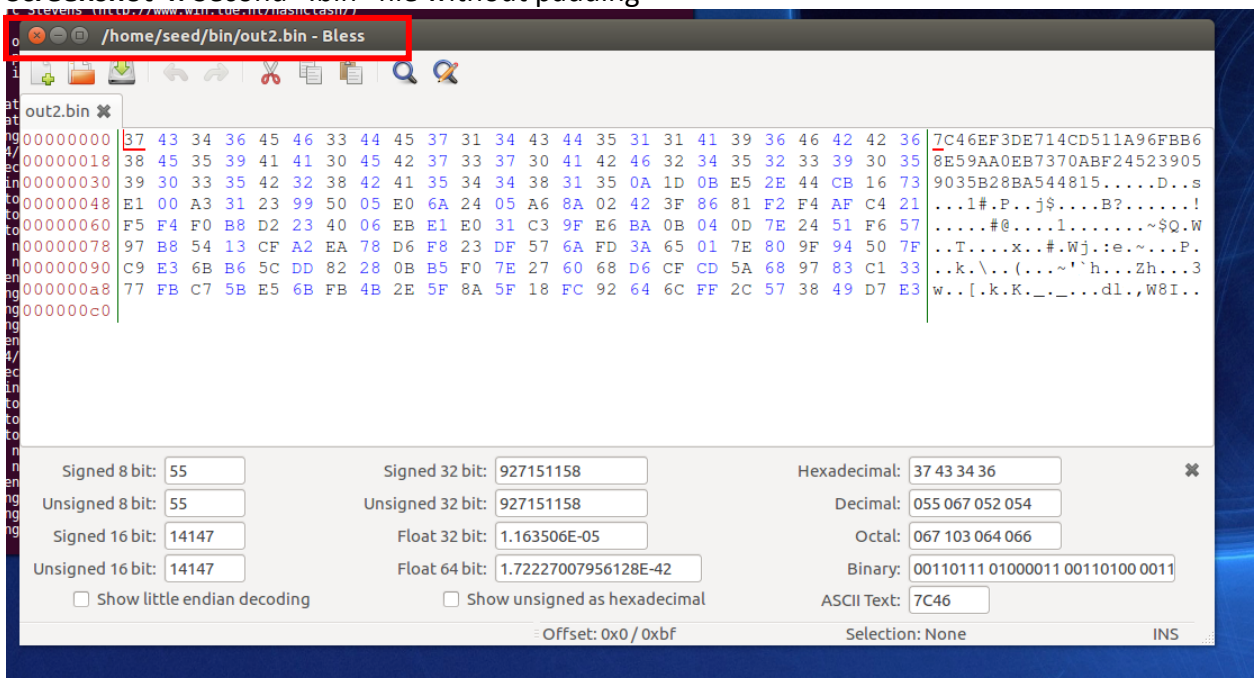
Upon running the collision tool with a 64 byte file, we see that the padding is no longer present. Please see the screenshots below for evidence of this.

**Screenshot 3:** First ".bin" file without padding



Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Screenshot 4:** Second ".bin" file without padding

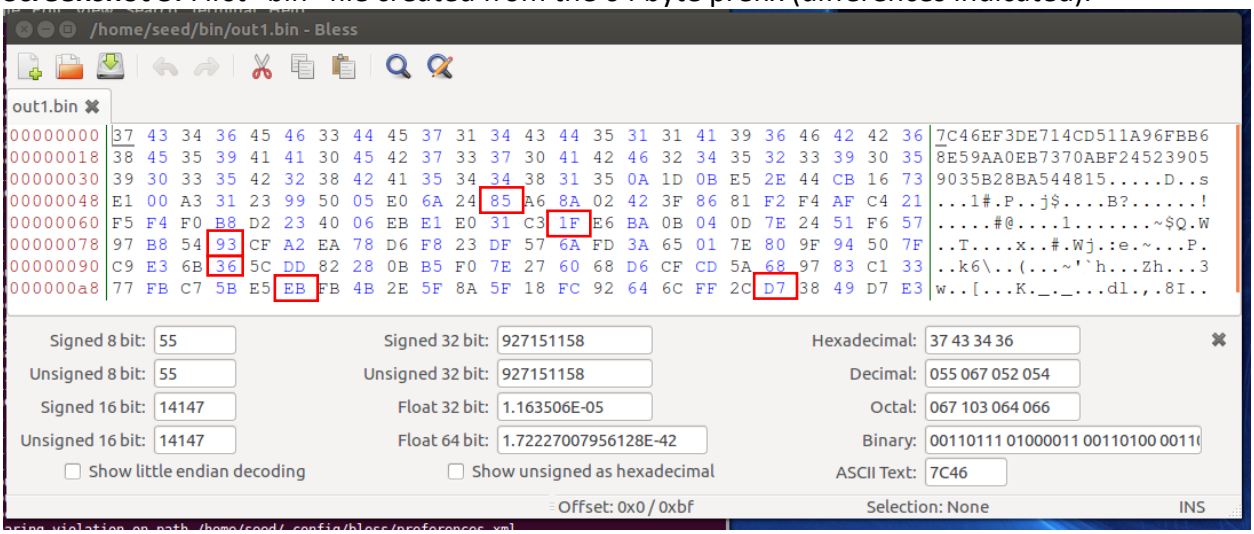



**Question 3.** Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.

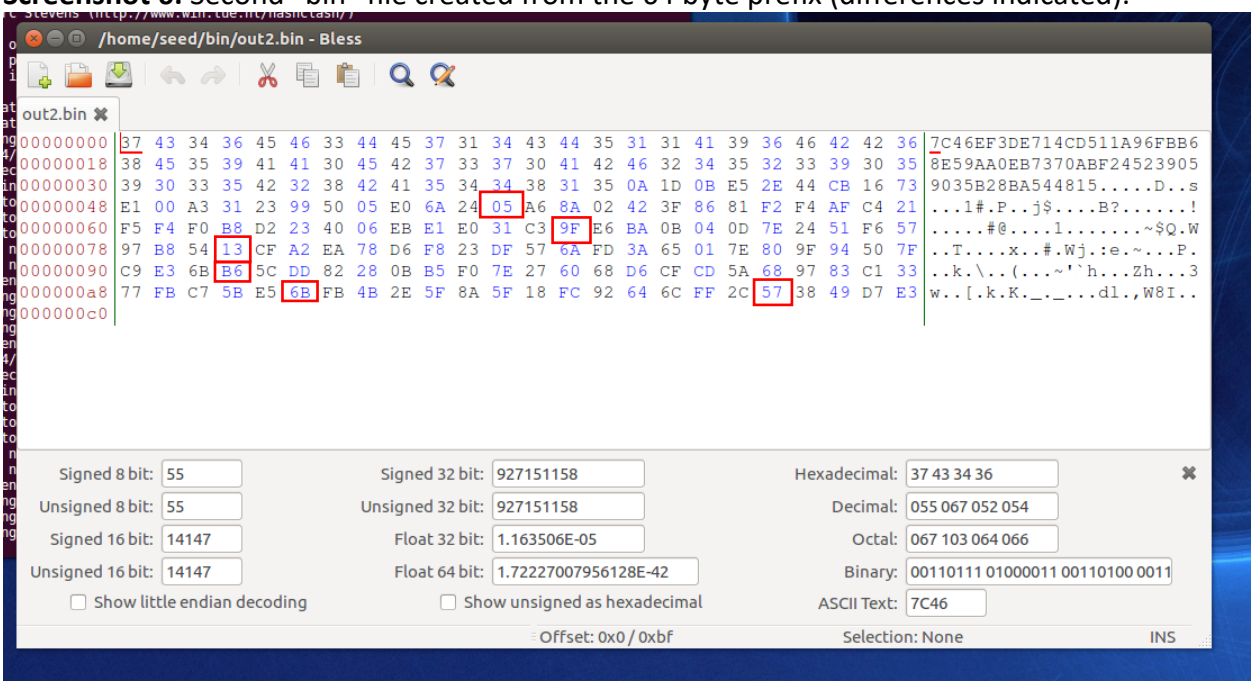
The two data files are not completely different. They share some common bytes, but there are several bytes which are different. These differences have been indicated in the following screenshots.

Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Screenshot 5:** First “bin” file created from the 64 byte prefix (differences indicated).




**Screenshot 6:** Second “bin” file created from the 64 byte prefix (differences indicated).



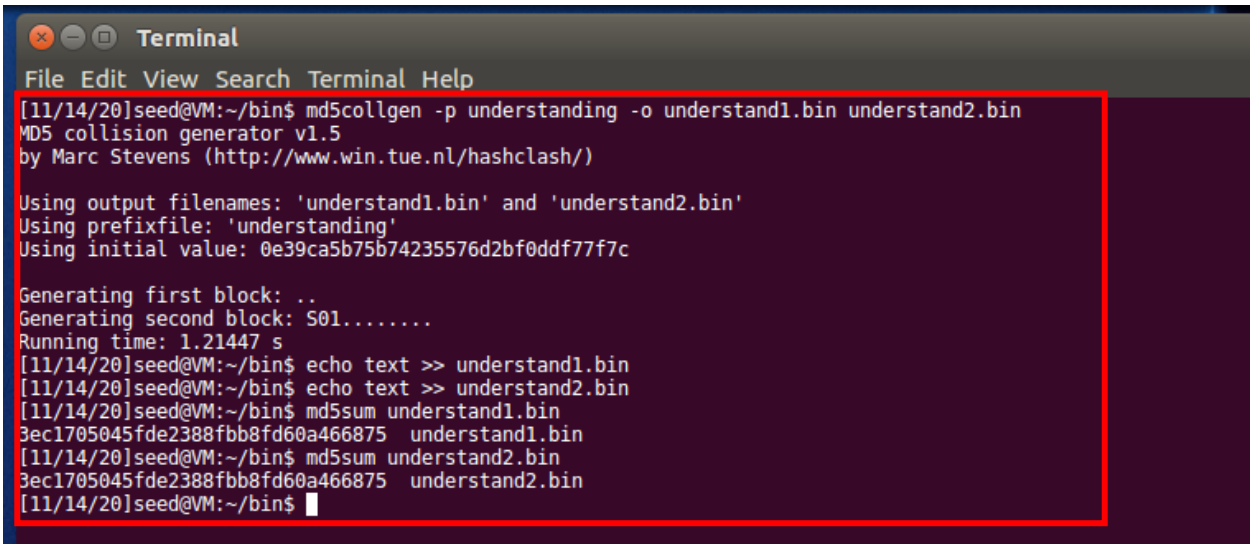
Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Task 2 – Understanding the MD5's Property**

To understand that adding the *exact same* suffix to a matching MD5 algorithm produces the same output, I performed the following steps:

1. Generated a 64 byte prefix (titled, "understanding")
2. Added text to both outputs (titled, understand1.bin and understand2.bin)
3. Compared the new hashes
4. Verified that they were the same

**Screenshot 7:** Displaying that the hashes were the same after adding the same suffix to both files



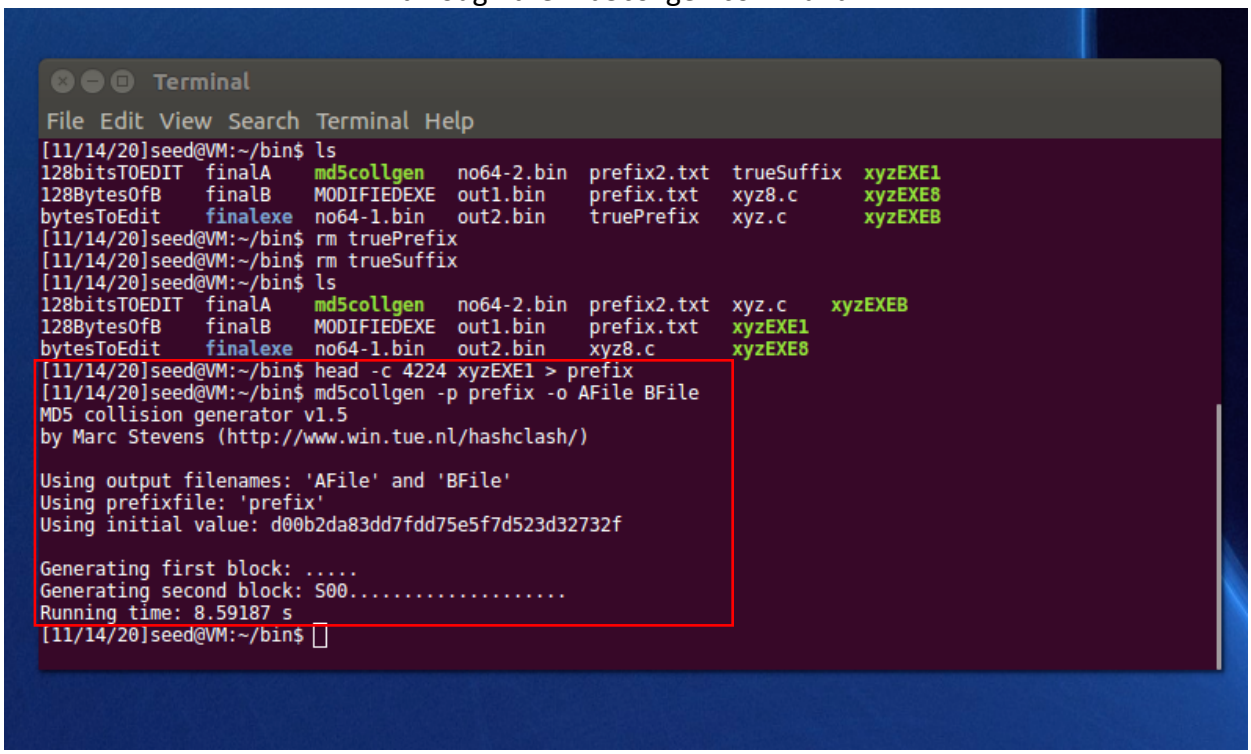

**Task 3 – Generating two executable files with the same MD5 Hash**

To make these two executable files with the same MD5 Hash, I performed the following steps:

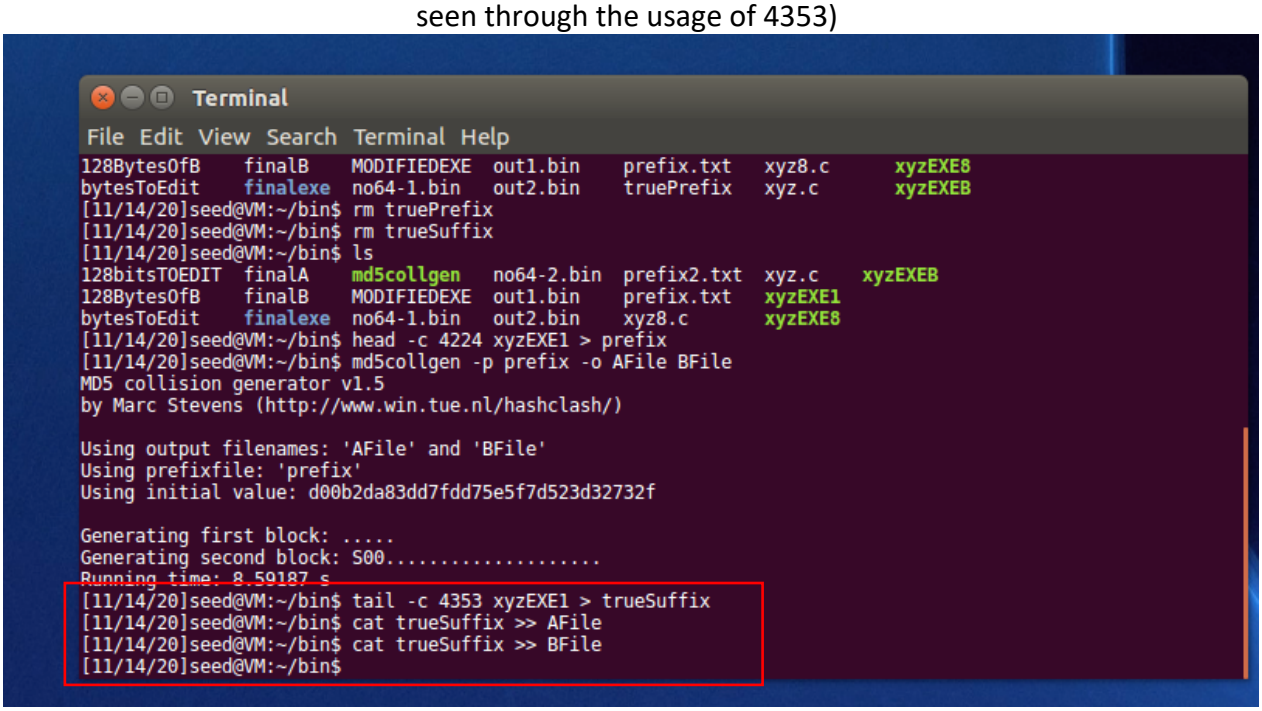
1. Created a program exactly the same as the one given in the instructions, except I put in 200 values of "A" so that I could easily find the values within the binary when I went to edit it
2. Crated the executable
3. Decided to use a value of 4224 (multiple of 64) to get the first 4224 bits of the executable for my prefix
4. Ran the md5collgen command to create the two binary files.
5. Left a 128 byte gap, and took the remaining bits. As a command, this looks like "tail -c 4353 [name of my executable] > stored in some file (I called mine "trueSuffix")
6. I added the suffix to both of the two binary files and compared their hashes noting that they were the same.
7. I also noted by using the "diff" function that the files were indeed determined to be different, even though they had the same hash.

Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Screenshot 8:** Getting the prefix of my executable (titled xyzEXE1), and making the two binaries through the md5collgen command

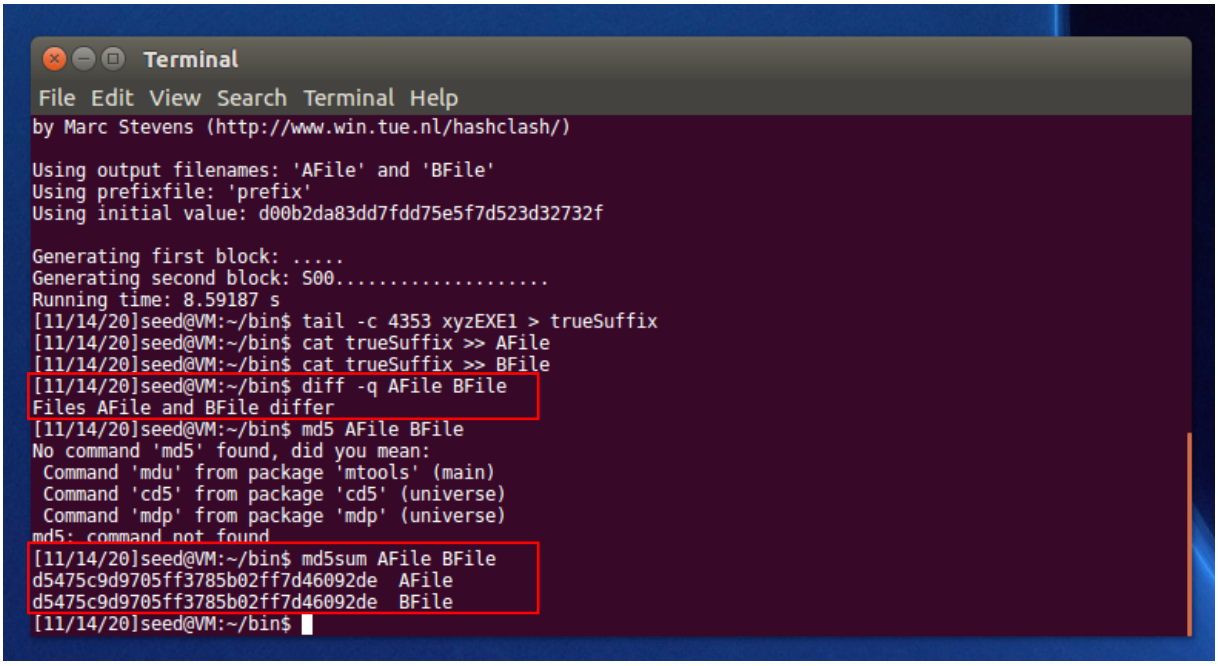



**Screenshot 9:** Getting the suffix to append to both of my binaries, leaving the 128 bit gap (as seen through the usage of 4353)

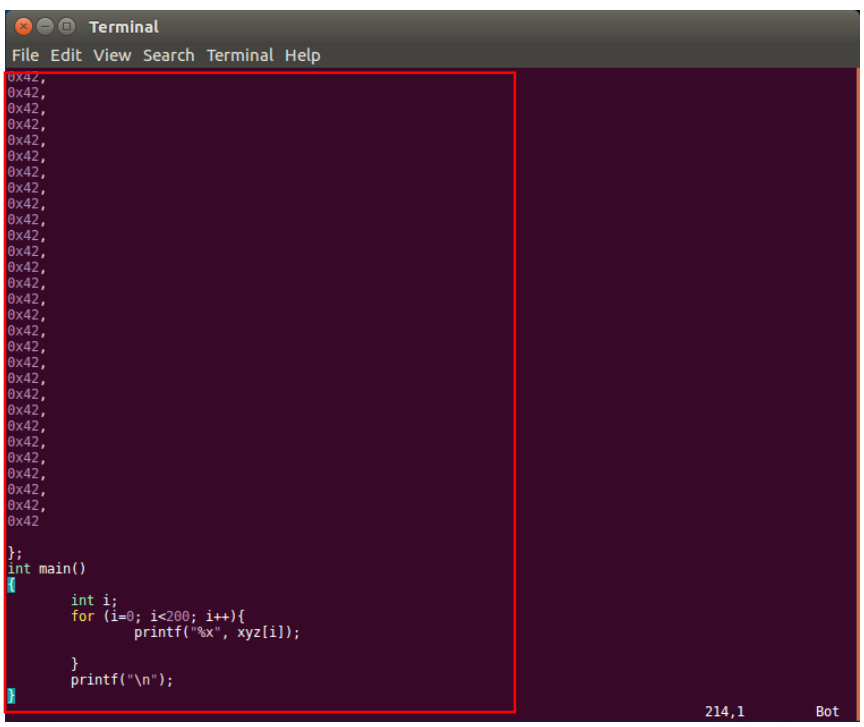


Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Screenshot 10:** Comparing if the two files differ via the "diff" command, but noting that the two files still share the same MD5 hash.




**Screenshot 11:** I realized after completing the lab that I had adjusted my original file executable to complete task 4, but this is the structure of my xyzEXE1 file, using "B's" instead of "A's" (I had tried to make the files match at the source code level but switched to adjusting binaries at the end instead). Also as a side note, there is a total of 200 "B's" that extends to the top of the file.



Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Task 4 – Making the two programs behave differently**

To complete this task, I performed the following steps:

1. Adjusted my original executable file to compare two arrays (as outlined within the lab document). I adjusted the executable so that if the arrays match, then it prints a message related to "safe" code, but if the arrays are different then prints a message for "malicious" code
2. I performed the exact same steps as task 3- obtaining my prefix, creating by two binaries from that prefix, and then creating my suffix to append to both binaries
3. However, this task is much trickier than task 3, as it has two arrays which we are looking to change. To do this, I noted that both of the binaries which were created from my prefix were missing the last 8 bytes of the first array. Hence, I took the first 8 bytes of the suffix file I had created and stored those in a separate variable, "remaining8Bytes".
4. I went ahead and completed the first array through adding this variable to both of the binaries.
5. Since both binaries now had the 8 bytes, this could be removed from the suffix I had created. Hence, I created a new variable to store the "tail" of the suffix file (tail -c +9 suffix > newSuffix).
6. Essentially, I was constructing both executables at the same time. The next step was to include the bytes which were between the first array and the second array (there was a gap between the two in the binary file). Hence, I created a variable "fillgap" to store the bytes between the two arrays.
7. I also created another variable to store all the contents after this "fillgap" in a separate variable (2ndarrWSuffix). This variable essentially holds the second array with the suffix.
8. Again, I continued to build out the executables, and added in the "fillgap" variable to both executables.
9. To make the final suffix, I took all the bytes after the second array in my 2ndarrWSuffix file and put that into a "FINALSUFFIX" file.
10. To construct the portion of the executable before the suffix, we can take the completed first array and store this in its own variable. That way, one of the files will have matching arrays and the other will have different arrays.
11. At this point, I had the prefix with both the arrays, the portion between them, as well as the final suffix. I ran both executables to see the results and noted that one produced a result for "good" code, and that the second produced the results for "malicious" code.

Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Screenshot 12:** edited C program that allows me to compare the two arrays (the arrays are defined at the top of the program with 200 entries each).

```
Terminal
File  Edit  View  Search  Terminal  Help
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41,
0x41
};

int main()
{
        int i;
        int sameNum = 1;

        for (i=0; i<200; i++) {
                printf("%x", xyz[i]);
                if (xyz[i] != newARR[i])
                        sameNum = 0;

        }

        if(sameNum)
                printf("SAFE CODE!");
        else
                printf("MALICIOUS CODE!");

        printf("\n");
}
                                                        428,1         Bot
```

**Screenshot 13:** Performing the same steps as for Task 3- creating the two binary files from the prefix and obtaining the suffix.

```
[11/14/20]seed@VM:~/bin/finalreport$ ls
a.out
[11/14/20]seed@VM:~/bin/finalreport$ head -c 4224 a.out > prefix
[11/14/20]seed@VM:~/bin/finalreport$ md5collgen -p prefix -o out1 out2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1' and 'out2'
Using prefixfile: 'prefix'
Using initial value: dc725d8947e74bcf81f931f389bd7672

Generating first block: .........
Generating second block: S01...
Running time: 10.3075 s
[11/14/20]seed@VM:~/bin/finalreport$ tail -c +4353 a.out > suffix
[11/14/20]seed@VM:~/bin/finalreport$
```

Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Screenshot 14:** Adding the missing 8 bytes from the first array to both binaries and removing those 8 bytes from the suffix that I was tracking

```
[11/14/20]seed@VM:~/bin/finalreport$ tail -c +4353 a.out > suffix
[11/14/20]seed@VM:~/bin/finalreport$ head -c 8 suffix > remaining8bytes
[11/14/20]seed@VM:~/bin/finalreport$ cat out1 remaining8bytes > out1WBytes
[11/14/20]seed@VM:~/bin/finalreport$ cat out2 remaining8bytes > out2WBytes
[11/14/20]seed@VM:~/bin/finalreport$ tail -c +9 suffix > newSuffix
[11/14/20]seed@VM:~/bin/finalreport$
```

**Screenshot 15:** Creating files for the gap between array 1 and array 2, as well as storing a file with the second array and the suffix. After doing this, I added the gap to both current "executables" (accidentally named them out1 and out 2 "Byes" but it should say "Bytes").

```
[11/14/20]seed@VM:~/bin/finalreport$ head -c 24 newSuffix > fillgap
[11/14/20]seed@VM:~/bin/finalreport$ tail -c +25 newSuffix > 2ndarrWSuffix
[11/14/20]seed@VM:~/bin/finalreport$ cat out1WBytes fillgap > out1ByesNGap
[11/14/20]seed@VM:~/bin/finalreport$ cat out2WBytes fillgap > out2ByesNGap
```

**Screenshot 16:** Creating the final suffix without any arrays, and obtaining the array which would be used to determine "good" or "malicious" code

```
alreport$ tail -c +201 newSuffix > FINALSUFFIX
[11/14/20]seed@VM:~/bin/finalreport$ tail -c +4161 out1WBytes > ArrayToAppend
```

**Screenshot 17:** Creating the two exec files and granting execute permissions

```
[11/14/20]seed@VM:~/bin/finalreport$ tail -c +4161 out1WBytes > ArrayToAppend
[11/14/20]seed@VM:~/bin/finalreport$ cat out1ByesNGap ArrayToAppend FINALSUFFIX > exec1
[11/14/20]seed@VM:~/bin/finalreport$ cat out2ByesNGap ArrayToAppend FINALSUFFIX > exec2
[11/14/20]seed@VM:~/bin/finalreport$ chmod +x exec1
[11/14/20]seed@VM:~/bin/finalreport$ chmod +x exec2
[11/14/20]seed@VM:~/bin/finalreport$
```

**Screenshot 18:** Running both files and getting the respective "SAFE CODE" and "MALICIOUS CODE" outputs

```
[11/14/20]seed@VM:~/bin/finalreport$ ./exec1
41414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141c212c7a2f6bd712a37aef50f2f421eef89d6584c2da46565e4196b7cac769ab2c6f8c8c9a56b05a93c76
b772b853c819774e5d9cadff87cb2795dc150403e2188620e9c1ffc7654b1d3334a6c4d2e33a61ee2d35aeeb8ec86e2e960ccc6da4
4e4bfd86f616f98f065ee6fdad0e5bfcaee6f3064d915ba60b04141414141414141SAFE CODE!
[11/14/20]seed@VM:~/bin/finalreport$ ./exec2
41414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141
41414141414141414141414141c212c7a2f6bd712a37aef50f2f421eef89de584c2da46565e4196b7cac769ab2c6f8c8c9a56b05a93c7e
b772b853c819774e5d9cadff8fcb2795dc150403e2188620e9c1ffc7654b1d3334a6c4dae33a61ee2d35aeeb8ec86e2e960ccc6da4
4e4bfd86f616f987065ee6fdad0e5bfcaee6f3064d9815ba60b04141414141414141MALICIOUS CODE!
[11/14/20]seed@VM:~/bin/finalreport$
```

Joseph Tsai
MD5 Collision Attack Lab Write-Up
November 14th, 2020

**Screenshot 19:** Comparing the MD5 hashes to ensure that they are the same

```
[11/14/20]seed@VM:~/bin/finalreport$ md5sum exec1
2d1447a007ed94894fd9f305fb8739b4  exec1
[11/14/20]seed@VM:~/bin/finalreport$ md5sum exec2
2d1447a007ed94894fd9f305fb8739b4  exec2
[11/14/20]seed@VM:~/bin/finalreport$
```