**MD5 Hash Collision Attack – Dorough**

**Task 1:**

* Created a prefix text file to embed into MD5 checksum binaries

Graphical user interface, text

Description automatically generated

* Used the *md5collgen* command to create executable binaries using the prefix.txt as a prefix for binary one and two
* Compared the checksums against each other to ensure they matched using the *diff* command
* Used the *xxd* command to dump the hexadecimal values of the binaries to visually check their contents

Graphical user interface

Description automatically generated

**Task 1 Question 1:**

The left-over bytes of data are set to 0s until the end of the prefix section.

**Task 1 Question 2:**

A picture containing text

Description automatically generated

The difference between 64-byte prefixes is that no 0s are needed to fill in the unused space.

**Task 1 Question 3:**

A picture containing text

Description automatically generated

Converting the binaries to a text file allows us to see the differences in bytes per line in each binary file. They are not wholly different. However, they do have notable differences.

**Task 2:**

When comparing the checksum of files. If they contain the same data, they will be compressed and encrypted in the exact same way.

Text

Description automatically generated

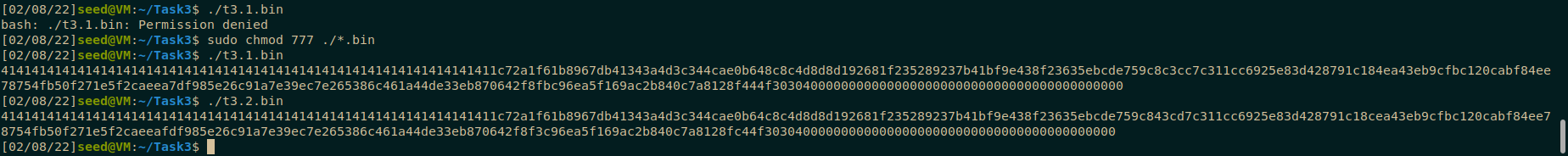
Here two file are created with identical contents. Even if both files are changed, as long as the contents are the same the md5 checksum will match between the files.

**Task 3:**

Here we take the prefix up to the array in the code that is divisible into a 64-byte block and copy all the bytes up to this point into a prefix file. Then we take the prefix plus a 128-byte offset and copy what’s left into the suffix file. From here we can make an md5 collision from the prefix into 2 new binary files. We take the last 128 bytes of the new binaries and save them as the p and q values. Concatenating them into a long file from prefix, p or q, and suffix, we get a matching md5 hash value with different file contents.

Graphical user interface

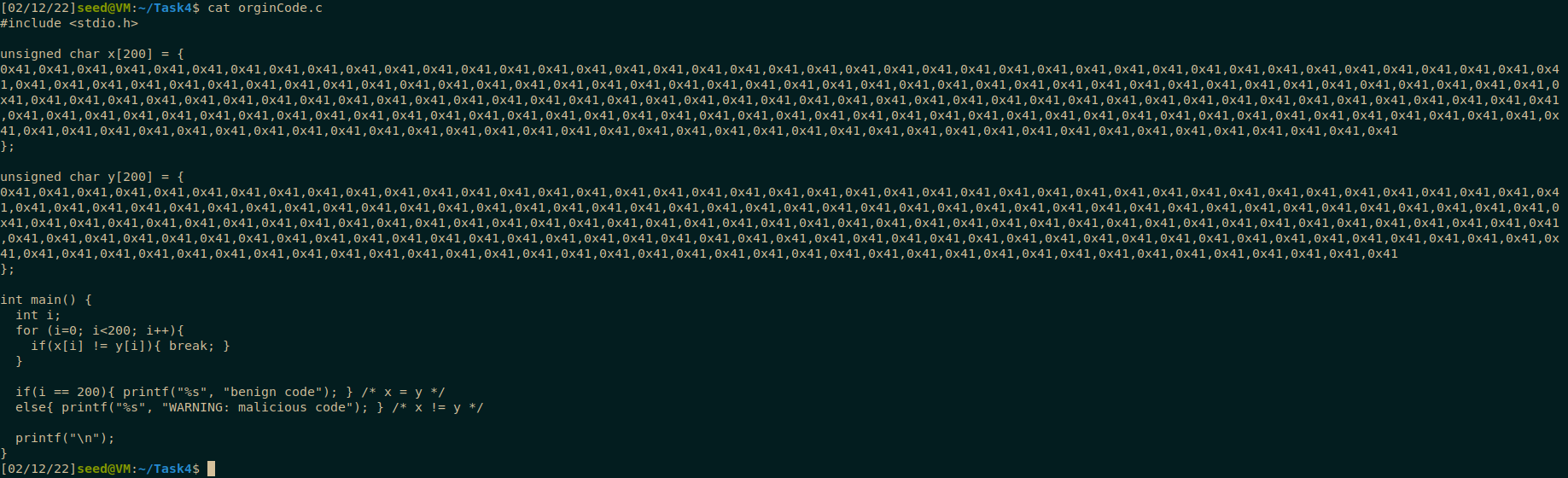
Description automatically generated



**Task 4:**

I have sat in my uncomfortable wooden LU dorm chair for over 6 hours straight counting individual bytes and aligning the arrays in the binaries until the binary code had the correct sized array to support the task. Here is the result of my fevered machine code campaign.

The first steps are the same as task 3. Create a program,

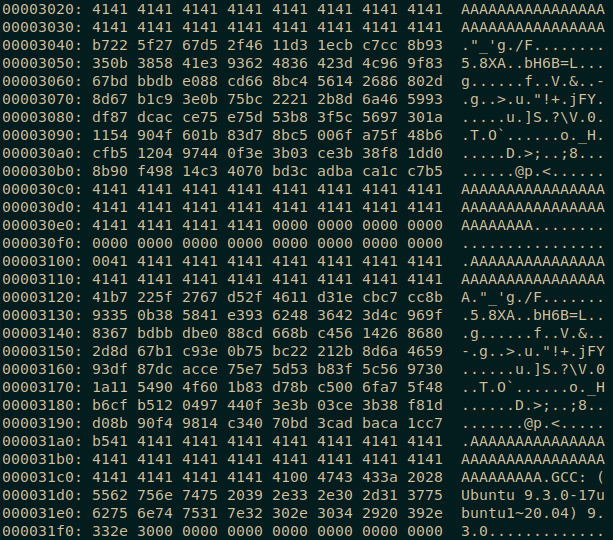


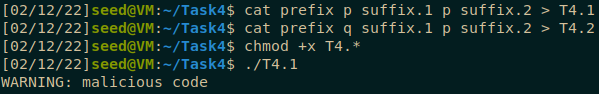
get the p and q values from a MD5 collision, and split a prefix and suffix to fit the values in an array. The difference in this task is that the code will have two arrays that must align. The format I used was:

[prefix] [p] [leftover first array and beginning of second array] [p] [rest of the code]

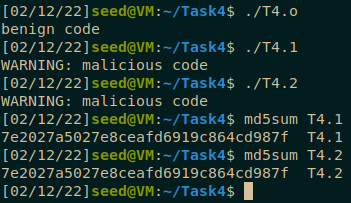
[prefix] [q] [leftover first array and beginning of second array] [p] [rest of the code]

What made this difficult was that the tools available all counted the bytes from different start values. So I had to print out the raw binary hexadecimal values and count each segment of the array to make sure the p and q values were inserted at the same offset.



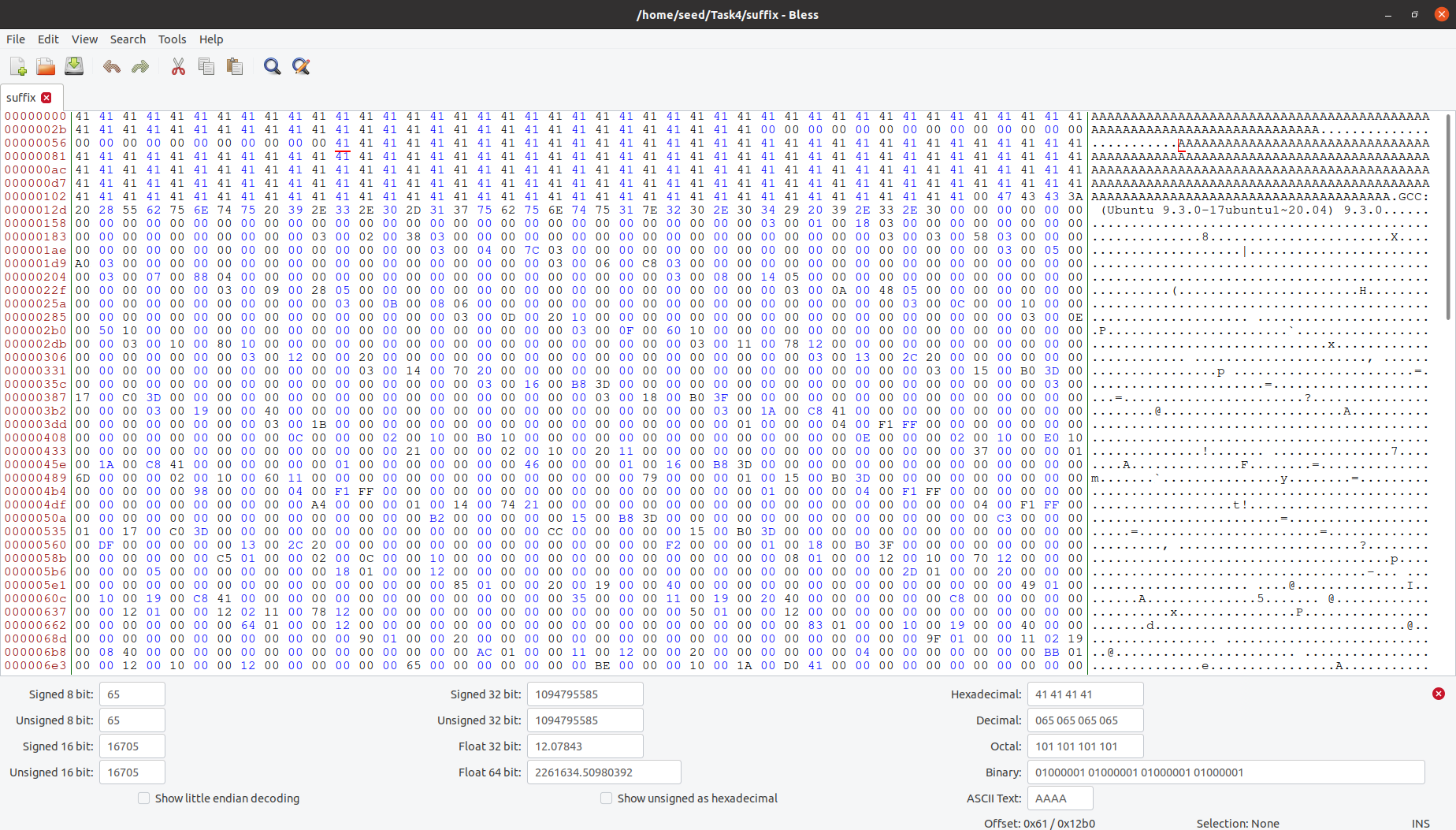
Here you see the second to last attempt to run the binary 

What the problem was is that the array should be 200 characters. Due to the tools starting at either 0 or 1 depending on the tool, the array ended up having an extra A at the end of the p or q value. You wouldn’t notice this unless you dissected each step and counted exactly how many characters were grabbed for the sections of code.



The size ended up being:

[32 As] [128 bytes of p or q collision] [40 As] = 200 characters. Which should be correct, but the final array size was 201 characters with an extra A on the end of both arrays.



In the end I carefully cut the extra A out of the second suffix and the code finally ran with matching MD5 hashes.

