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

This task requires me to generate two different files that have the same MD5 hash values. The way we do this is by making the beginning parts of the two files the same (share the same prefix) and then use the MD5 Collision Generator (md5collgen) command to generate these two output files which we will apply the MD5 algorithm to and get the same hash for both files. See Figure 1 for how the MD5 Collision Generator works (Source: Dr. Du's Lab Instructions).

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

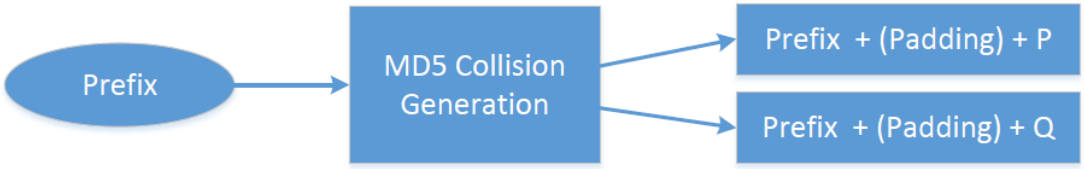



Figure 1: MD5 collision generation from a prefix

Figure 2 (below) shows the execution of this task. First I created the shared prefix, then I ran the md5collgen command using that prefix and designated two output binary files.

```
Terminal
[10/29/19]JCogswell@Attacker-131:~$ echo "hello from CYSE330" > prefix_classdemo
[10/29/19]JCogswell@Attacker-131:~$ md5collgen -p prefix_classdemo -o out1_class
.bin out2_class.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1_class.bin' and 'out2_class.bin'
Using prefixfile: 'prefix_classdemo'
Using initial value: 2ba3dadf3bfe2558e8e2c41ccce3d81e

Generating first block: .....
Generating second block: S01.......
Running time: 6.91345 s
[10/29/19]JCogswell@Attacker-131:~$ diff out1_class.bin out2_class.bin
Binary files out1_class.bin and out2_class.bin differ
[10/29/19]JCogswell@Attacker-131:~$ sha256sum out1_class.bin
bd92a16f4c1071aa102faff116ee9df6fa17316d9276e85d19144e97648b5974  out1_class.bin
[10/29/19]JCogswell@Attacker-131:~$ sha256sum out2_class.bin
3df0752a334c33b94494cd6e737f8cc9bdc9bc41594be0c4187d0bf996c45d79  out2_class.bin
[10/29/19]JCogswell@Attacker-131:~$ md5sum out1_class.bin
58ddaf87a20c3780b471b7d9d5f16a45  out1_class.bin
[10/29/19]JCogswell@Attacker-131:~$ md5sum out2_class.bin
58ddaf87a20c3780b471b7d9d5f16a45  out2_class.bin
[10/29/19]JCogswell@Attacker-131:~$
```

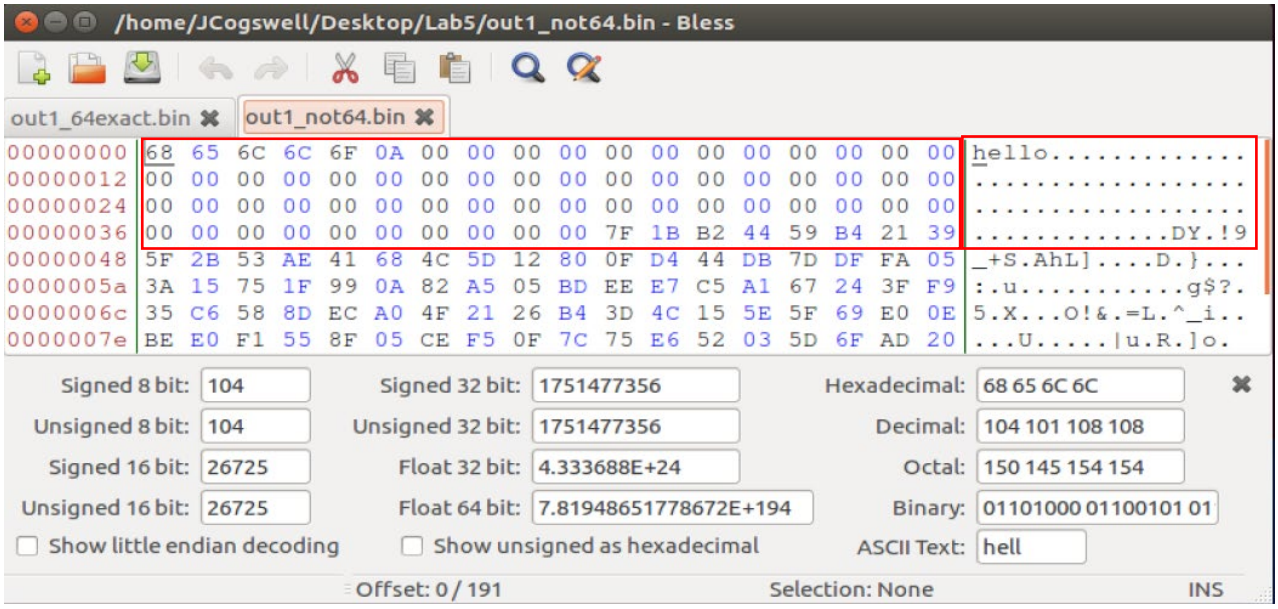
*Figure 2*

**Observations / Thoughts:**

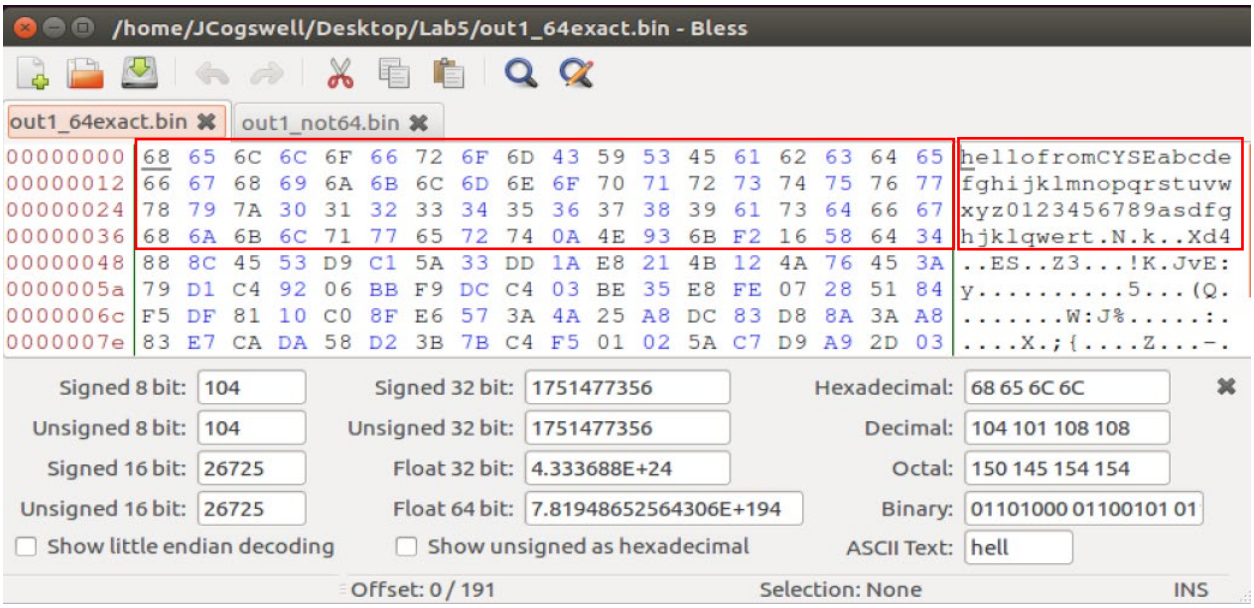
- Notice that when using the diff command to compare out1_class.bin and out2_class.bin we see that they are different.
- Using SHA-256 hashing algorithm, we get different hashes for each file. Using MD5 produces the same hash for each file.

**Questions:**

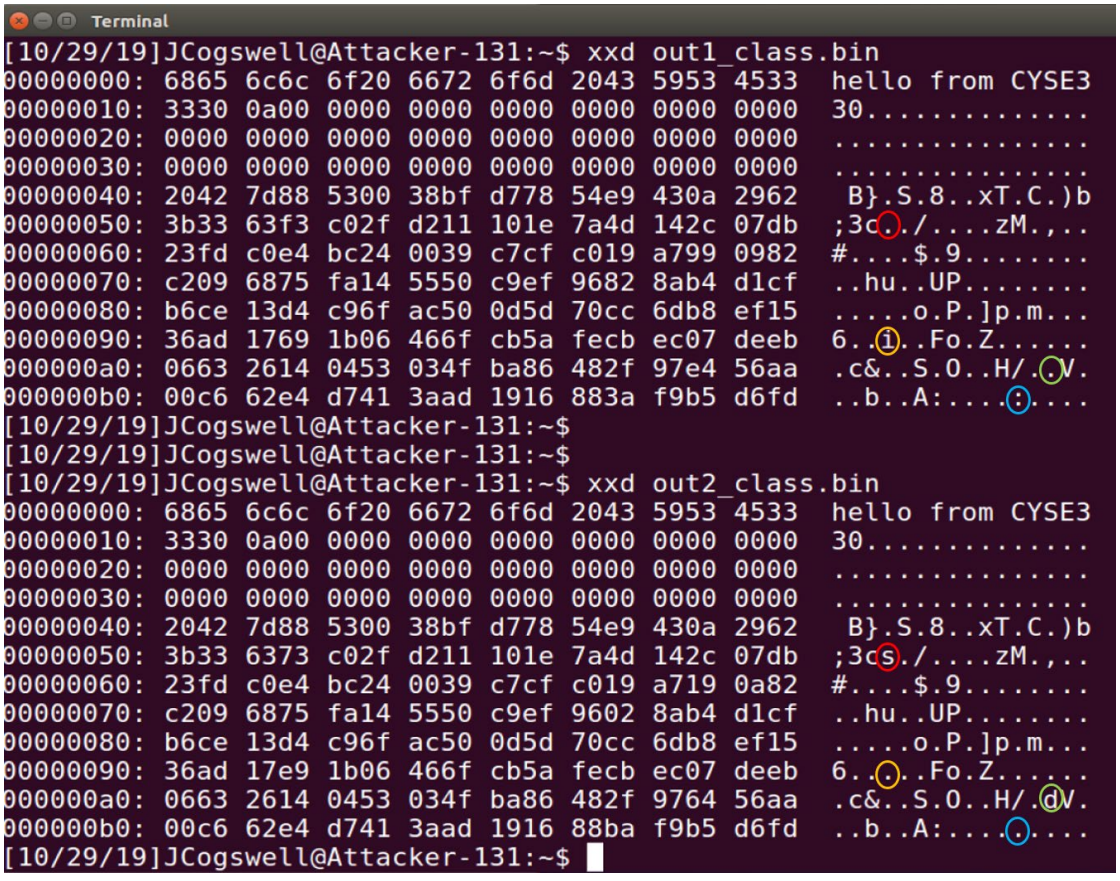
1. **If the length of your prefix file is not a multiple of 64, what is going to happen?**
   a. Since MD5 requires sizes to be multiples of 64 bytes each, then if your prefix is not a multiple of 64 bytes, the rest of the prefix will be padded with zeros to meet the 64 byte requirement. See the screenshot below that demonstrates this.



2. **Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.**
    a. If the prefix file is exactly 64 bytes, then no padding will be added. See the screenshot below.




3. **Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.**
    a. No they're not. The figure below shows a few of the differences in the two files, just to make it clear that they're similar but not identical. The colors are there to show the locations of the changes between the files.



**Task 2: Understanding MD5's Property**

This task helps me understand some of the MD5 algorithm properties which are important for the tasks to follow. MD5 divides input data into blocks of 64 bytes and then computes the hash iteratively on these blocks. The MD5 compression function takes two inputs, a 64-byte data block and the outcome of the previous iteration. This compression function produces a 128-bit IHV (intermediate hash value) and then this value is fed into the next iteration. For the last iteration, the IHV is the final hash value. The *first* IHV ($IHV_0$) input is a fixed value. Figure 3 explains how the MD5 algorithm works.

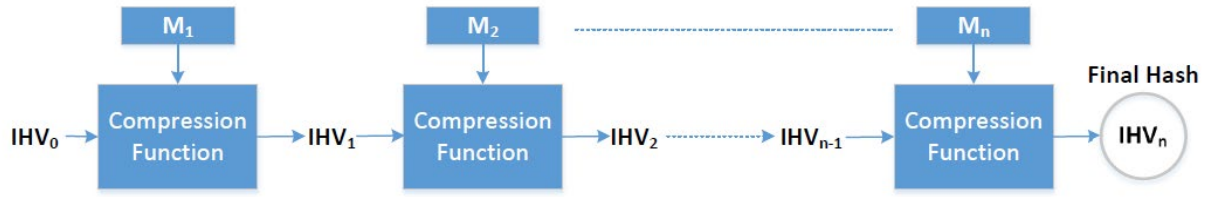



*Figure 3*

From the lab instructions: based on how MD5 works, we can derive the following property of the MD5 algorithm: Given two inputs *M* and *N*, if *MD5(M) = MD5(N)*, i.e., the MD5 hashes of M and N are the same, then for any input *T*, *MD5(M || T) = MD5(N || T)*, where || represents concatenation. This means that if the inputs M and N have the same hash, adding the same suffix T to them will result in two outputs that have the same hash value. This holds not only for the MD5 algorithm, but also for many other hash algorithms. Now, in this task I am designing an experiment to demonstrate that this property holds for MD5. Figure 4 shows that when I concatenate out1_class.bin/out2_class.bin with suffix.txt and create a new file from their outputs and then apply the MD5 hashing algorithm, they will still produce the same hash (blue boxes), even though the files differ (red box).

```
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ cat out1_class.bin suffix.txt > out
1_class_long.bin
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ cat out2_class.bin suffix.txt > out
2_class_long.bin
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ diff out1_class.bin out2_class_long
.bin
Binary files out1_class.bin and out2_class_long.bin differ
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ md5sum out1_class_long.bin
b683efdb6675f4d127ec9d951a2ae3f8  out1_class_long.bin
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ md5sum out2_class_long.bin
b683efdb6675f4d127ec9d951a2ae3f8  out2_class_long.bin
```

*Figure 4*

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

In this task I am provided with a program that generates two different executable files that will have the same MD5 hash values. This program fills an array with 400 'A' characters so it's easy to find the location of the characters in the Bless hex editor. I then choose a 128-bit region inside the array of As making sure that the prefix (everything before the starting point of my region) is a multiple of 64 bytes and the suffix starts the next byte after the region ends and consists of the rest of the program. See Figure 5 for an illustration for how this works.

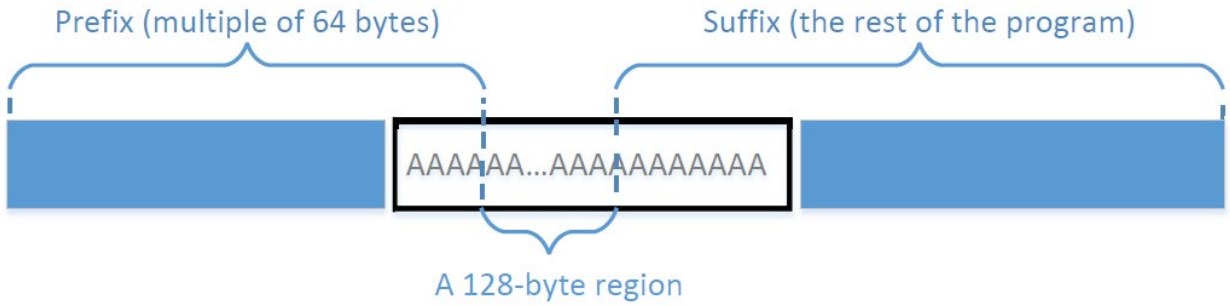



*Figure 5*

Next, I run the md5collgen on the prefix to generate two outputs that have the same MD5 hash value. I am letting P and Q represent the second part of these outputs (each having 128 bytes). Then we are testing the same property mentioned before → *MD5 (prefix || P) = MD5 (prefix || Q)*.

Based on the property of MD5, we know that if we append the same suffix to the above two outputs, the resultant data will also have the same hash value. Basically, the following is true for any suffix:

*MD5 (prefix || P || suffix) = MD5 (prefix ||Q || suffix)*

Therefore, we just use P and Q to replace 128 bytes of the array (between the two dividing points), and we will be able to create two binary programs that have the same hash value. Their outcomes are different, because they each print out their own arrays, which have different contents. We use the Bless hex editor to find the location of the array so we can create the prefix and suffix. Figure 6 shows the Bless hex editor and the location of the array of As. Notice the offset value is 4160 – we will use that to create the prefix by defining the header as the first 4160 bytes of the file.

Figure 7 shows the creation of the Q, P, the prefix, the suffix, and the md5collgen command being executed. Finally, as also shown in Figure 7, we concatenate the prefix and suffix to P and Q and generate the two programs from those outputs, respectively. Then the MD5 algorithm is applied to the two newly generated programs (a1_class.out and a2_class.out).

**Note:** The *tail -c +4289 t3 > suffix_class* simply means that the suffix is the last portion of the *t3* program, starting at byte 4289. 4289 (suffix) is just the 4160 (prefix end byte) + 129 (because 128 bytes belong to the region we are creating as shown in Figure 5).

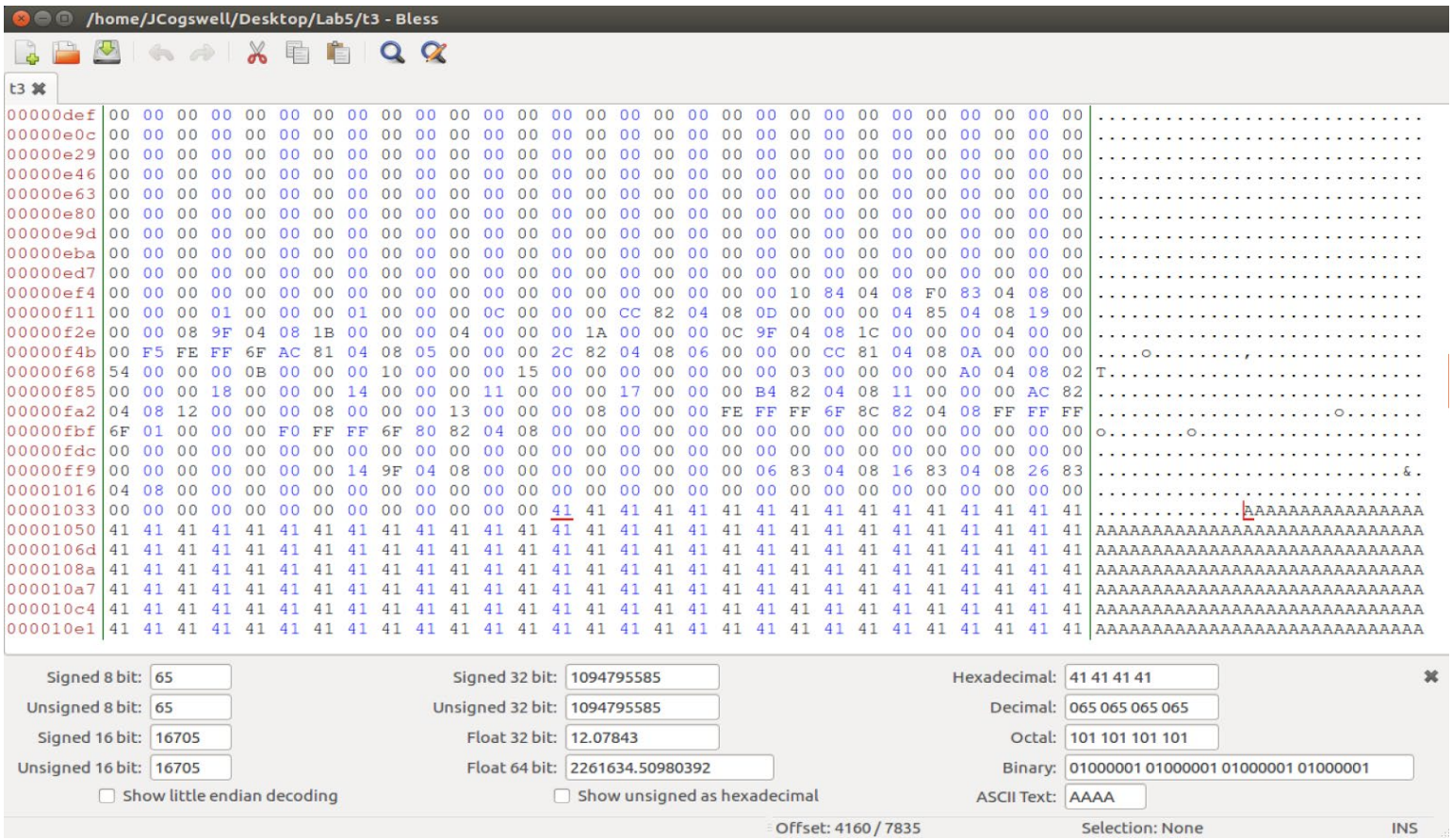



*Figure 6*

*Figure 7*

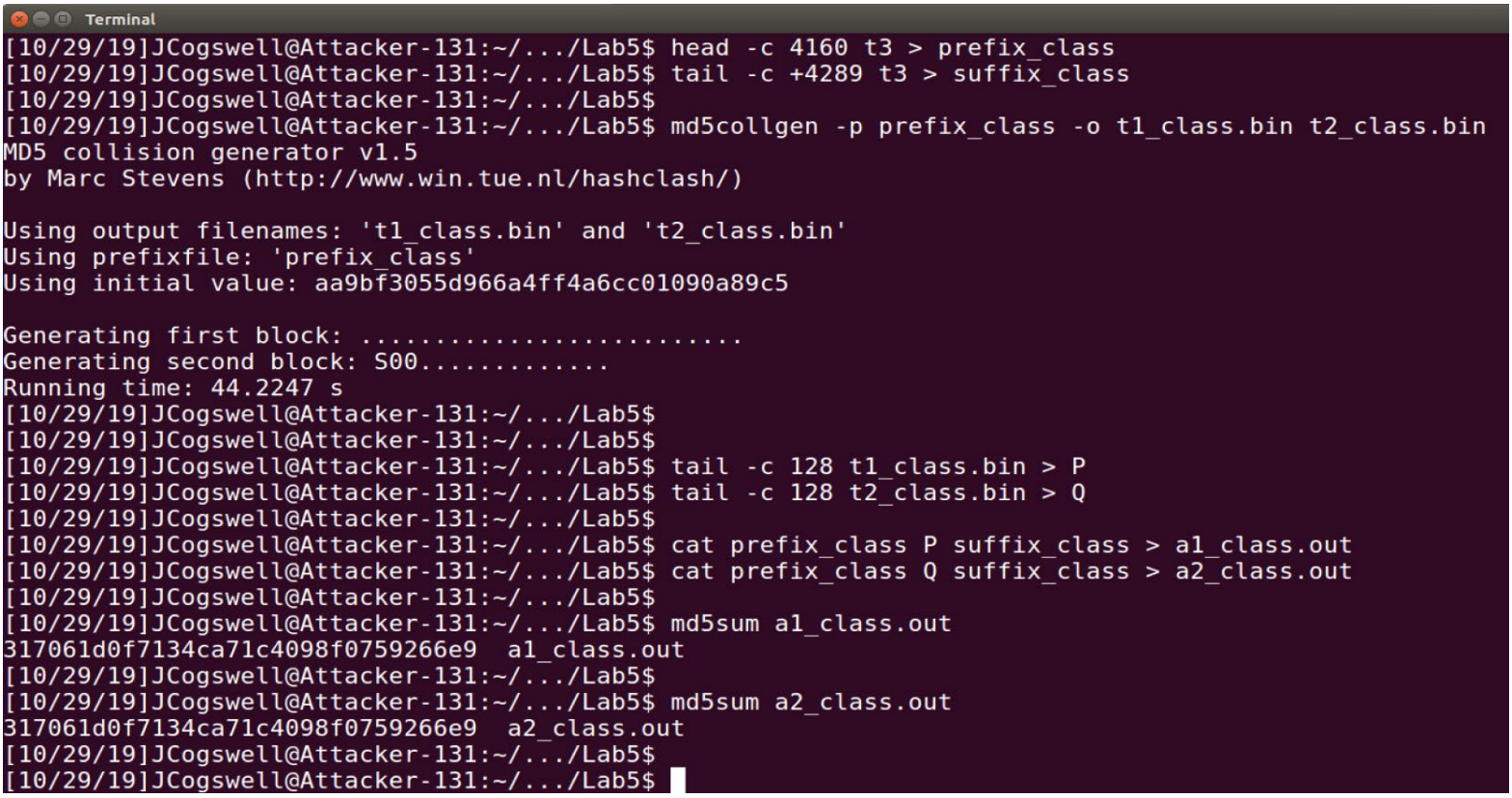


Figure 8 shows the execution of the two programs, that the two programs are different (red box), and marks a few of the differences since they're not easy to find.

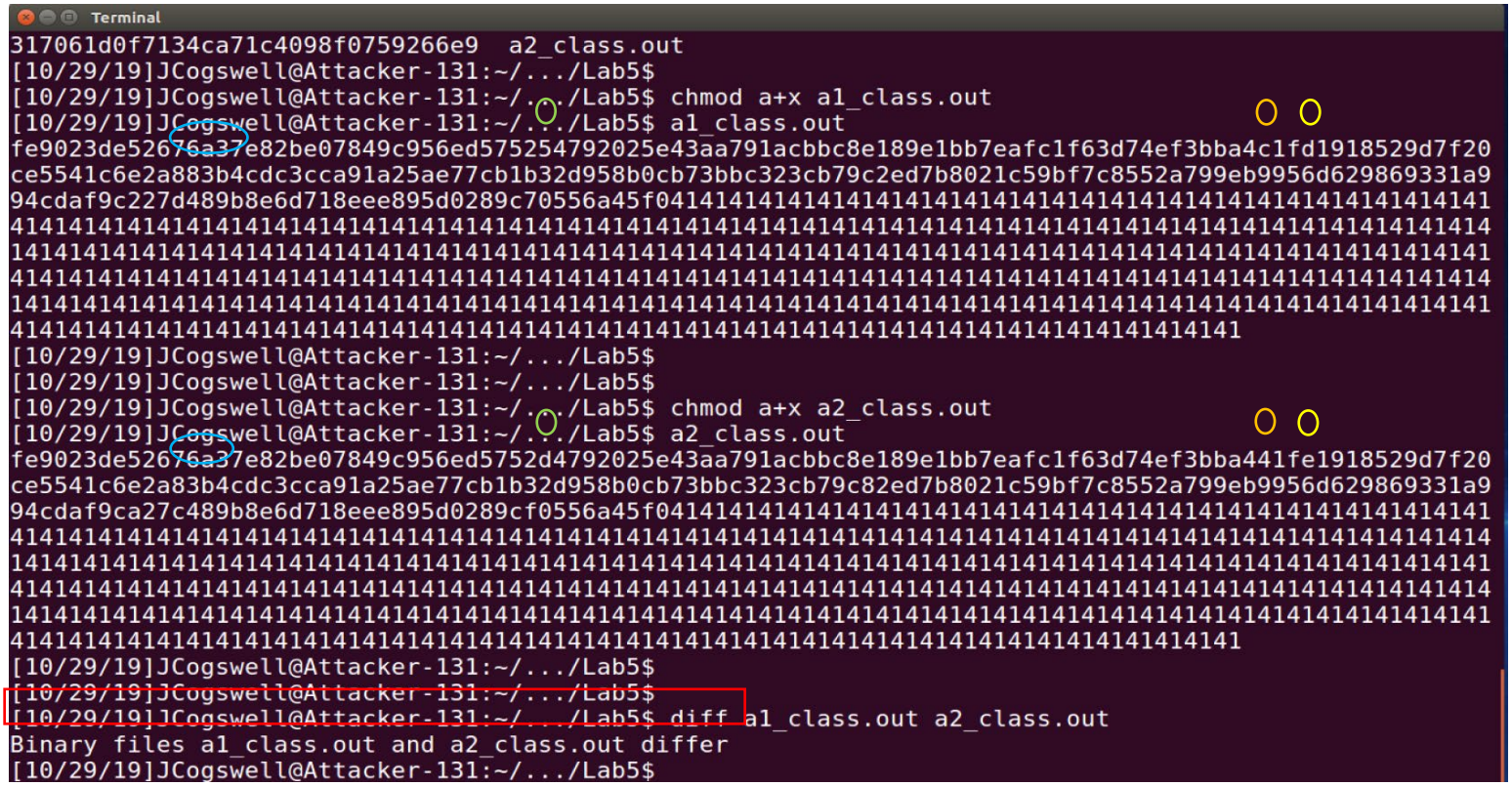



*Figure 8*

**Task 4: Making the Two Programs Behave Differently**

In the previous task, we created two programs that have the same MD5 has but their behaviors are different. However, their differences are only in the data they print out; they still execute the same sequence of instructions. In this task, we would like to achieve something more significant and more meaningful.

In this task, we pretend that I have created a program that does something good and I am able to get a certificate for it signed by the authority. Now I want to get my malicious program also signed by the certificate authority but since they certainly wont sign a certificate for a malicious program, I exploit this MD5 collision attack to generate the same MD5 hash so that when the authority checks the hash, the certificate will be valid for the malicious program since the hashes are the same. Now I've obtained a valid certificate for both my good program and my malicious program. I will launch this attack and show that one of my programs will execute benign instructions and my other program will execute malicious code.

To do this, we initialize the arrays X and Y in my program with some values that help us find their locations in the executable binary file. I need to change the contents of the two arrays so we can generate two different versions that have the same MD5 hash. In one version, the contents of X and Y are the same, so the benign code is executed; in the other version, the contents of X and Y are different, so the malicious code is executed. Figure 9 illustrates what the two programs should look like.

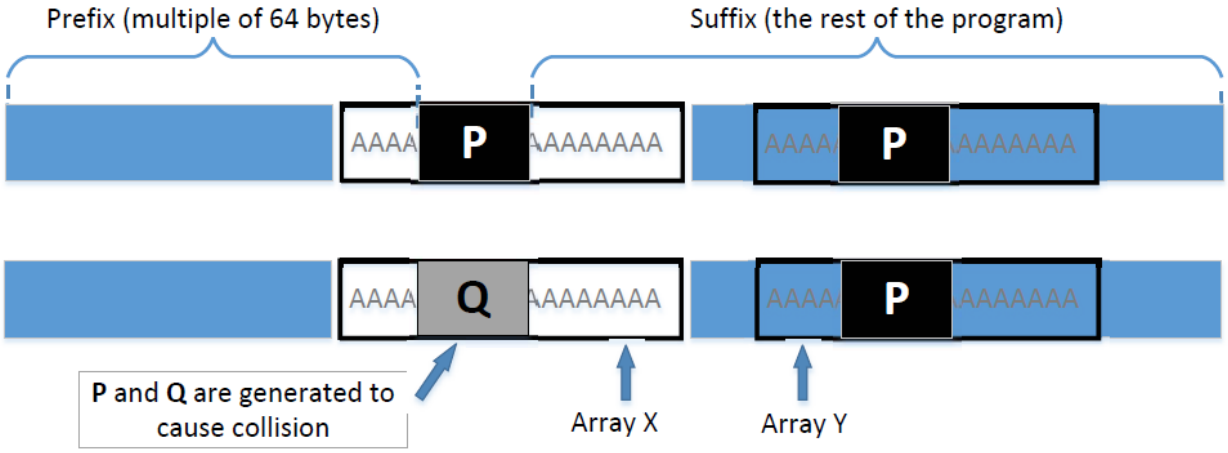



*Figure 9*

From above, we know that these two binary files have the same MD5 hash value, as long as P and Q are generated accordingly. In the first version, the contents of X & Y are the same, while in the second version, we make the contents different. Therefore, the only thing we need to change is the contents of these two arrays, and there is no need to change the logic of the programs. Figure 10 shows the Bless hex editor for the X array. Figure 11 shows the Bless hex editor for the Y array. Figure 12 shows the creation of the non-malicious program (a1_class.out) using the locations of the X array. Figure 13 shows the creation of the malicious program (a2_class.out) using the locations of the Y array. Figure 14 shows the execution of the attack and the resulting execution of benign code from a1_class.out and the malicious code from a2_class.out.

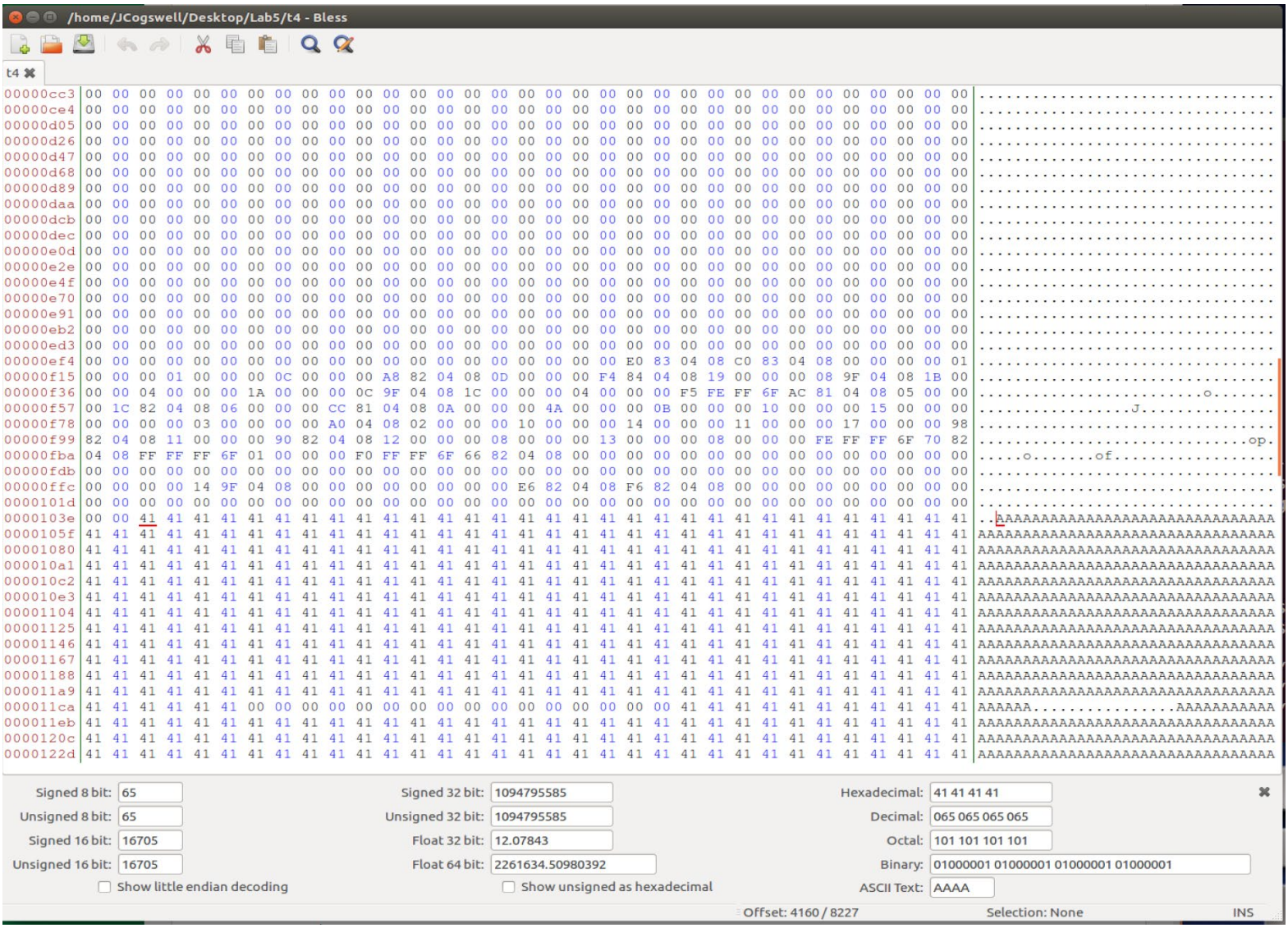



*Figure 10 (X array – Location for header: 4160)*

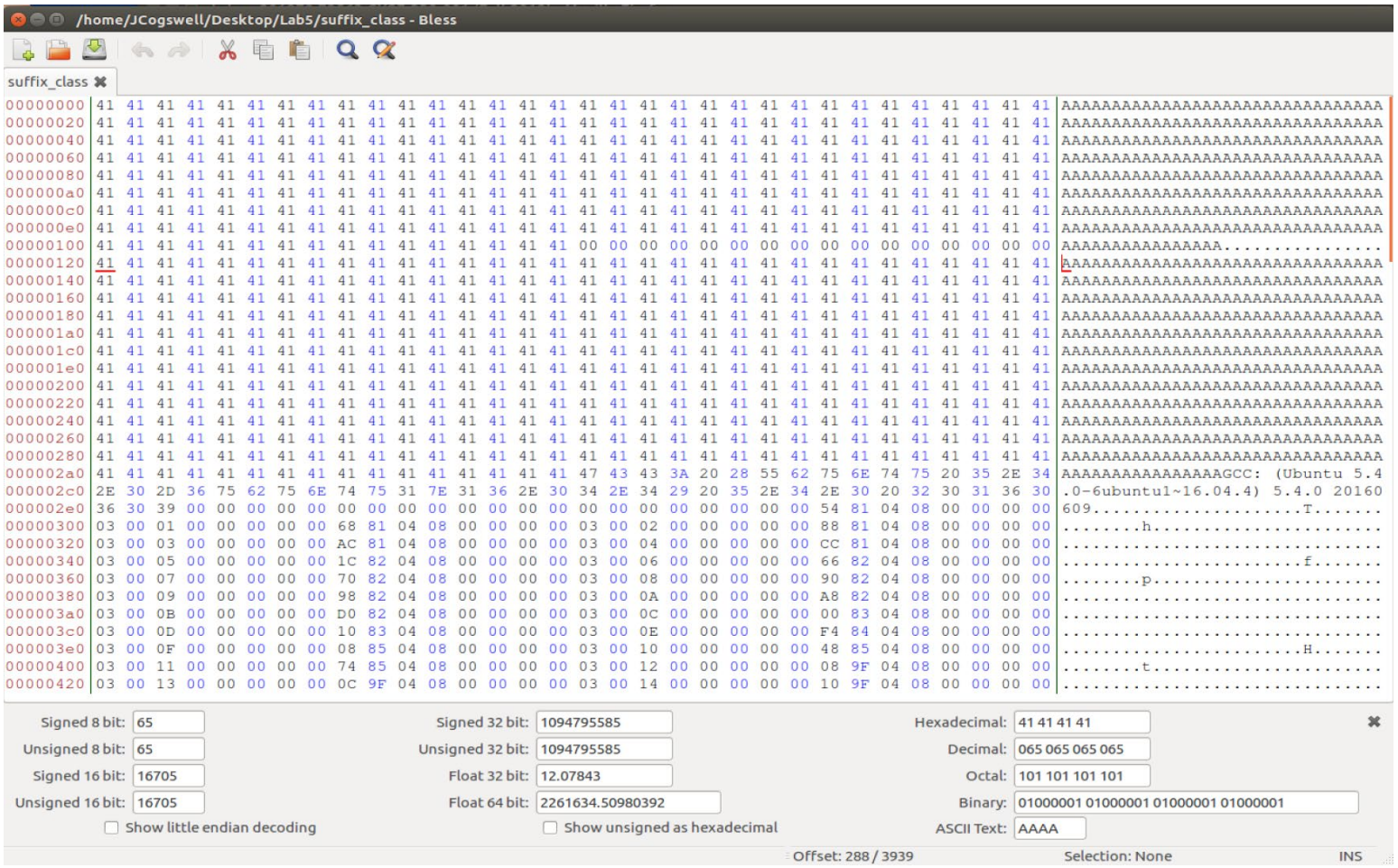



*Figure 11 (Y array – Location for header: 288)*

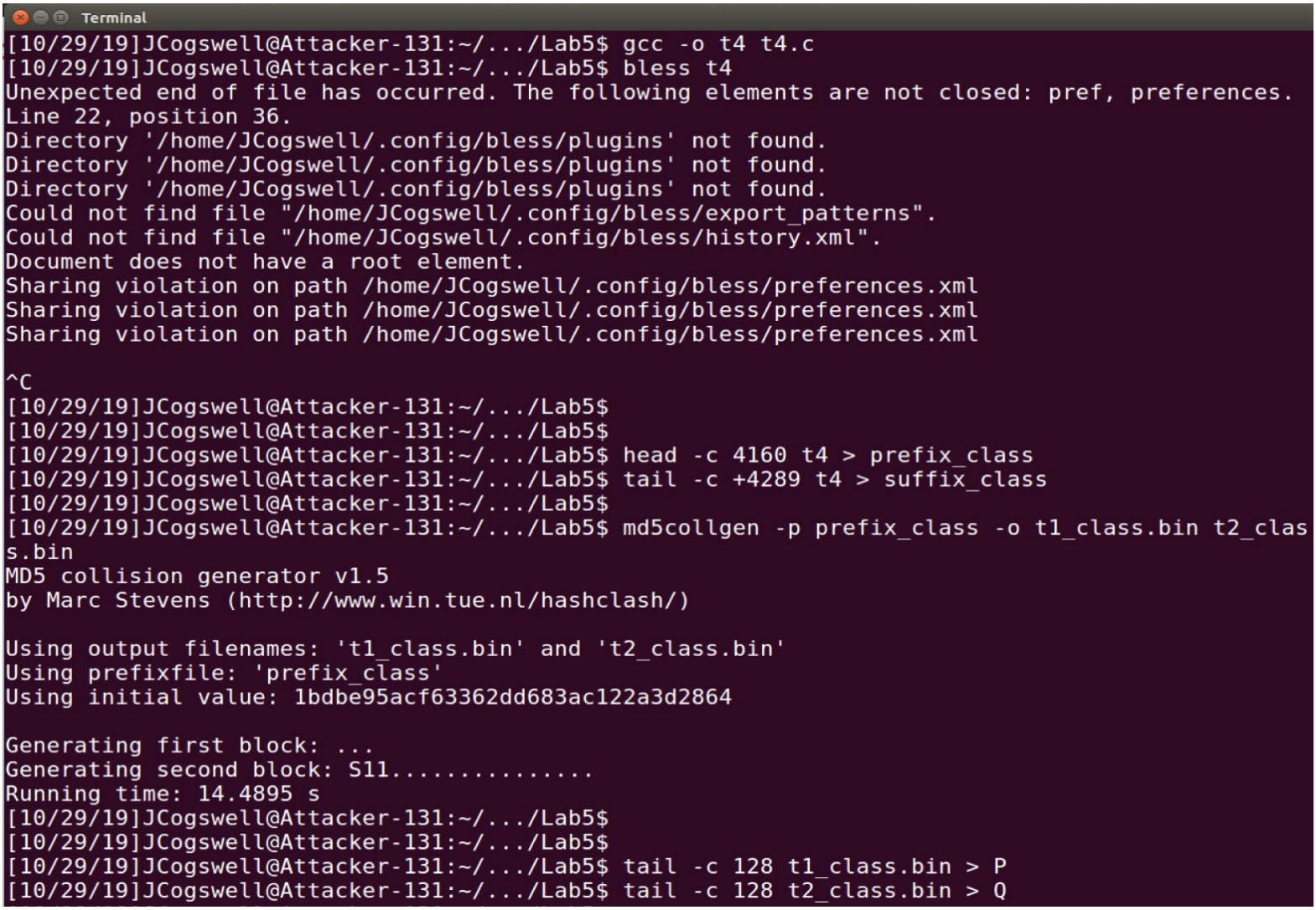

Terminal
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ gcc -o t4 t4.c
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ bless t4
Unexpected end of file has occurred. The following elements are not closed: pref, preferences.
Line 22, position 36.
Directory '/home/JCogswell/.config/bless/plugins' not found.
Directory '/home/JCogswell/.config/bless/plugins' not found.
Directory '/home/JCogswell/.config/bless/plugins' not found.
Could not find file "/home/JCogswell/.config/bless/export_patterns".
Could not find file "/home/JCogswell/.config/bless/history.xml".
Document does not have a root element.
Sharing violation on path /home/JCogswell/.config/bless/preferences.xml
Sharing violation on path /home/JCogswell/.config/bless/preferences.xml
Sharing violation on path /home/JCogswell/.config/bless/preferences.xml
^C
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ head -c 4160 t4 > prefix_class
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ tail -c +4289 t4 > suffix_class
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ md5collgen -p prefix_class -o t1_class.bin t2_clas
s.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
Using output filenames: 't1_class.bin' and 't2_class.bin'
Using prefixfile: 'prefix_class'
Using initial value: 1bdbe95acf63362dd683ac122a3d2864
Generating first block: ...
Generating second block: S11.............
Running time: 14.4895 s
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ tail -c 128 t1_class.bin > P
[10/29/19]JCogswell@Attacker-131:~/.../Lab5$ tail -c 128 t2_class.bin > Q

*Figure 12*

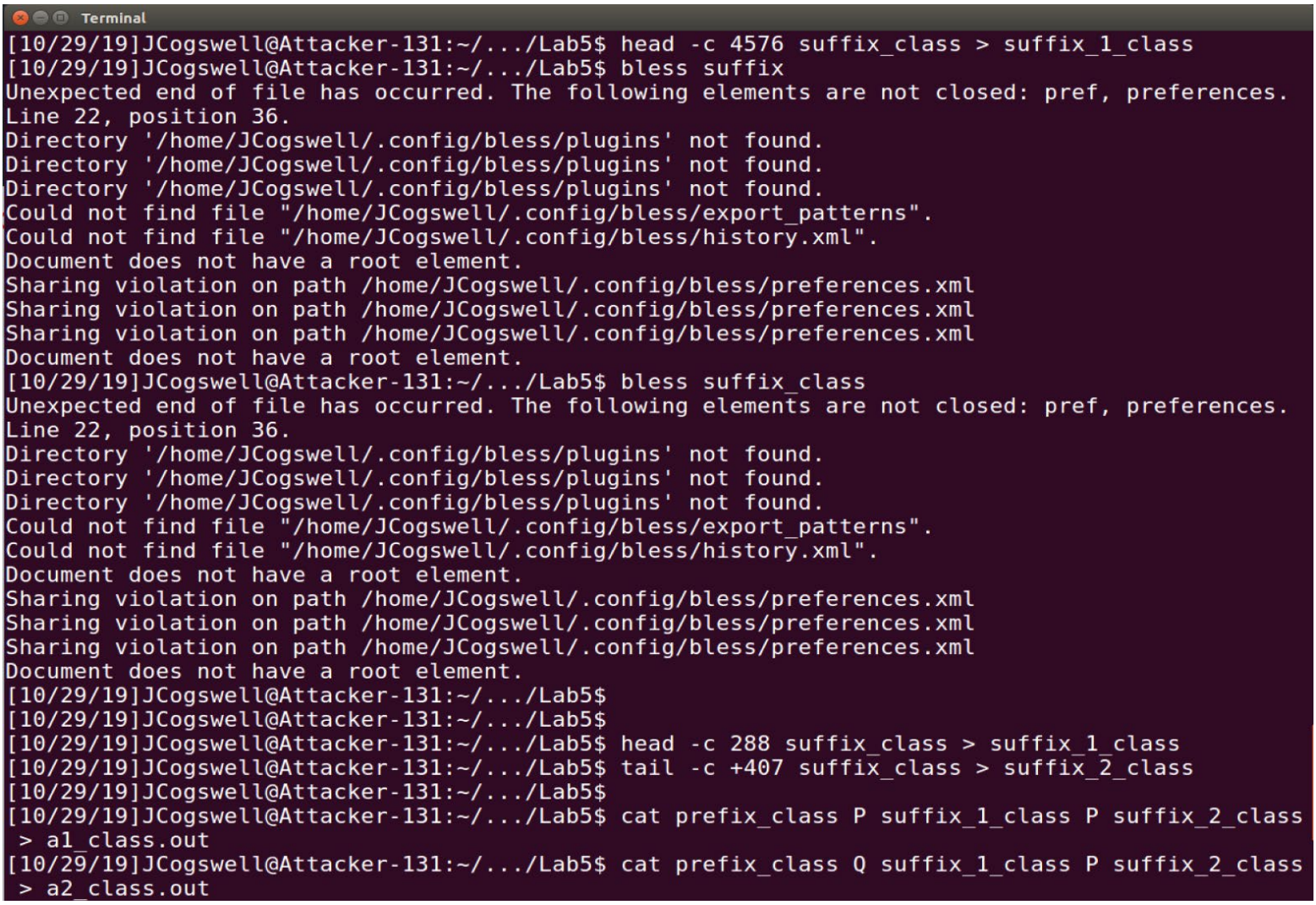



*Figure 13*

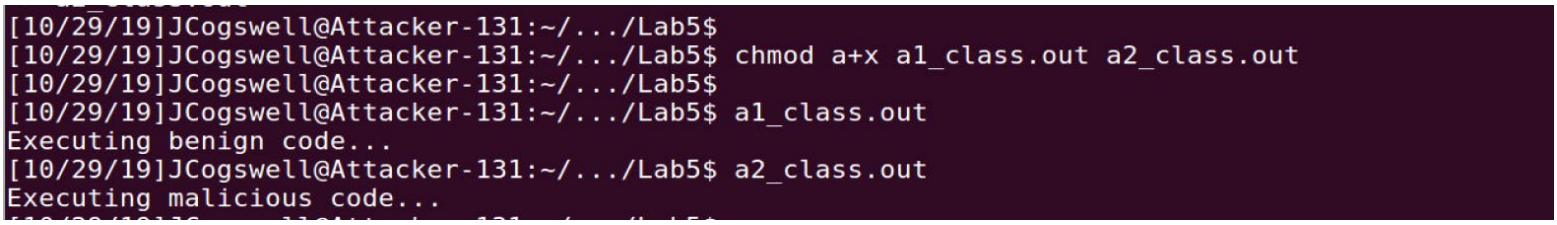



*Figure 14*