**INTRODUCTION TO INFORMATION SECURITY**

**LAB 3**

**PRANAV MURALI**

**A20555824**

**Task 2.1: Generating Two Different Files with the Same MD5 Hash**

We first create a text file titled “collision attack lab” which is named prefix.txt

A screenshot of a computer

Description automatically generated

We create two distinct output files, out1.bin, and out2.bin, and create the output blocks using the MD5 collision generator function.

A computer screen shot of a program

Description automatically generated

A screenshot of a computer

Description automatically generated

Using the diff program, we are able to verify whether the output files vary from one another.

A screenshot of a computer

Description automatically generated

To inspect the contents of out1.bin and out2.bin, respectively, we use the xxd command.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

We use the md5 checksum to find the hash of the output files

A screenshot of a computer

Description automatically generated

First, we create a file with 11 and 24 bits for the first question. If the text file is not a multiple of 64, the remaining bits are padded to zeroes.   
To address the second query, we generate a file of precisely 64 bytes. As we can see below, this file does not have zero padding.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Regarding the third question, we are unable to claim with certainty that the data produced by the md5collgen for the output files output1.bin and output2.bin differs substantially. Although there are byte changes across the output files, they are generally similar.

A computer screen shot of a computer code

Description automatically generated

We use the ***bless*** editor to check hashes of output filesA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Here, we can use the diff command to view the contents of the output files and compare the byte differences for the same.

A screenshot of a computer program

Description automatically generated

**Task 2.2: Understanding MD5’s Property**

A screenshot of a computer

Description automatically generated

We verify that the newly joined files, f1, and f2, have a valid MD5 checksum after which we examine the text files f1.txt and f2.txt's MD5 checksum.

A screenshot of a computer

Description automatically generated

We can observe that the separate files f1.txt and f2.txt have the same value, as does the combined file f1 and f2. This demonstrates unequivocally that a new hash is generated upon merging the files, distinct from the hash of the individual files.

**Task 2.3: Generating Two Executable Files with the Same MD5 Hash**

We create a file with 200 bits of 0x41 using the command given below

A screenshot of a computer program

Description automatically generated

Using nano function, we can edit text

A computer screen shot of a program

Description automatically generated

We use the ***gcc*** command to append the code to another file task3.o

A computer screen shot of a computer

Description automatically generated

We append the head by *4224 bits* view the md5collgen data for the output files of task3.o

**A computer screen with white text

Description automatically generated**

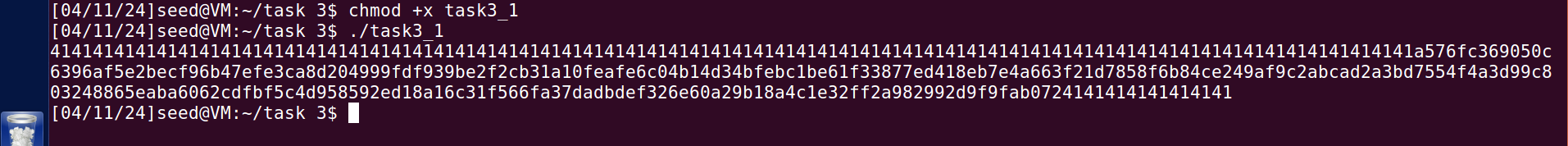
We append *4352 bits* to the tail of task 3. o file and create a suffix. Then, we append the tail by *128 bits* for the output files task3\_a.bin and task3\_b.bin.

We then combine the prefix and suffix files to form two new files and compare their hash values using md5 checksum. We see that the hash values are the same.

A screenshot of a computer program

Description automatically generated

To inspect the output, we open the final output file. The bits between As, which are appended and represent our needed output, are visible to us.



**Task 2.4: Making the Two Programs Behave Differently**

Like in the previous task, we add and save our code using the nano editor. We utilize the characters 0x43 for both arrays in the arrays. Next, we append the code to a file called task4.o using the gcc command.

**A computer screen with text on it

Description automatically generated**

A screenshot of a computer

Description automatically generated

Two additional files, suffix\_4 and prefix\_4, are created by appending the head and tail of task4.o, which have 4160 and 4288 bits, respectively. To view the data for task4.o output files task4\_a.bin and task4\_b.bin, respectively, we use the md5collgen program.

A computer screen with white text

Description automatically generated

Using the bless editor, we inspect the hash value of suffix\_4 and note the spacing between the bits.

A computer screen with a computer code

Description automatically generated

To create two files, p and q, we append the 128-bit tail of the output files task4\_a.bin and task4\_b.bin. The md5 checksums for task4\_a.bin and task4\_b.bin are then compared, and they both appear to be the same. Additionally, we note that files p and q have the same permissions.

A screenshot of a computer

Description automatically generated

To create two new files, presuffix\_4 and postsuffix\_4, we add the 192-bit head and 320-bit tail of suffix\_4.   
To create the necessary final task4final and task4malware files, we join the output files task4\_a.bin and task4\_b.bin with the presuffix\_4 and postsuffix\_4 files.

A screenshot of a computer program

Description automatically generated

We obtain the expected result after running the task4final and task4malware files.



A black background with white text

Description automatically generated