

## **MD5 Hashing Algorithm Collision Attack Lab Report**

Hayden Eubanks

School of Business, Liberty University

CSIS 463-B01

Dr. De Queiroz

August 31, 2023

## **MD5 Hashing Algorithm Collision Attack Lab Report**

### **Introduction:**

Hash functions are an extremely useful form of cryptography that allows for the mathematical validation of input data without revealing the underlying data that is being validated (Microsoft, 2022). This feature can seek to provide data confidentiality and integrity, but also non-repudiation as hashing can be implemented to validate communications (Basta, 2018). Hashing algorithms are unique from other forms of encryption in that the encryption performed is not reversible and the encrypted signature is the element used for verification (Amazon, 2023). Further, hashing algorithms differ from symmetric and asymmetric encryption by encrypting data into cyphertext values of fixed size regardless of the input plaintext size (Long, 2019). While this feature is integral to the functioning of hashing algorithms, it also leads to the potential for collisions where two distinct input values map to the same hashed value (Shen et al., 2022). Several hashing algorithms that were once widely used have now been deprecated due to their susceptibility to collisions including the Message-Digest 5 (MD5) and the Secure Hash 1 (SHA1) algorithms, and in this lab collision attacks on MD5 will be examined as it is especially susceptible to collision attacks (Shen et al., 2022). Through exploring the vulnerabilities present within MD5 a greater understanding of secure hashing algorithms can be achieved and security professionals can be better equipped to mitigate vulnerabilities associated with hashing functions.

The MD5 algorithm seeks to encrypt data through five data modification steps. These steps include the padding of bits, appending the message length, initializing the message digest buffer, processing the input data, and returning a hashed value (Mohammed Ali &

Kadhim Farhan, 2020). The tasks in this lab highlight the importance of the values associated with the steps above and in doing so seeks to explore the effects of modifying the number of bytes passed to the algorithm, the effect of varying message lengths, and the transformation of the input data itself. This then extends to exploring the weaknesses of MD5 and gaining an understanding of the risks of implementing it today. While MD5 has been deprecated for modern use, exploring collisions within MD5 will allow a security professional to gain a better understanding of the workings of hashing algorithms and better understand the risks associated with weak implementations.

To accomplish this understanding, the lab begins by providing an introduction to the way that MD5 processes data and the importance of processing block size in this process. With this, the `md5collgen` function is used to generate two differing values that will form a collision when hashed with MD5. The output values of this function are then examined on the byte level and contrasted to gain insight into what data values change. This information is then built upon to establish how this vulnerability in MD5 could lead to malicious code passing as non-malicious code. After this, the lab practically establishes the fundamental concept that appending the same value to two identically hashed values will maintain hash congruency. This opens the possibility for examining code and modifying particular sections so that the hash value remains the same despite the code being fundamentally altered. Through these exercises, the lab seeks to highlight the importance of understanding vulnerabilities associated with hashing algorithms, and a security professional could use this knowledge to implement algorithms that minimize collisions and in doing so increase security.

**Lab Procedure:**

To begin this lab, the md5collgen function must first be installed ([Screenshot1](#)) and added to the system's executable files ([Screenshot2](#)) allowing the function to be used on the system. This function is a collision generator for MD5 taking an input value and returning a second output value that has the same MD5 hash signature but differing by several bytes from the original message. While the md5collgen function is unique to the tests performed in this lab, it aims to represent the potential for collisions, and the security implications of these collisions are explored in later lab tasks.

With the prerequisite steps then accomplished, the lab work can begin by generating a new prefix file ([Screenshot3](#)) and then passing this file to the md5collgen function to generate a collision with this file. The diff function can be used to verify that differences exist between the files ([Screenshot5](#)) but when the file hashes are checked with the md5sum function ([Screenshot6](#)) it can be seen that the hash values are identical. This highlights the problem at the core of the MD5 collision attack which is the ability for distinct values to hash to the same hash sum (Obaida et al., 2022). As hash values are irreversible (Alenezi, Alabdulrazzaq, & Mohammad, 2020) there is then no way for an observer of the hash value to identify which of the possible inputs for that hash value was passed to MD5. Further, and most relevant for this lab, there is also no way to verify if two given hash values truly originated from the same plaintext. This is a major security concern as it is then possible for malicious code to hold the same hash signature as benign code (Mohammed Ali & Kadhim Farhan, 2020). This can be verified in the lab by using the bless command to observe the binary executable files generated by md5collgen ([Screenshot7](#)) and noting the differences that exist between them ([Screenshot8](#)).

The first section of this lab then asks several questions and through exploring these questions, a better understanding of MD5 can be obtained. The first of these questions prompts the exploration of modifying the input file size to multiples of 64 bytes. MD5 processes data in 64-byte blocks (Alenezi et al., 2020) and if the input data is not a multiple of 64 bytes long, then extra zeros are appended until a multiple is reached. This can be verified by creating a file of 64 characters, each being a byte long ([Screenshot9](#)), and then passing that file to md5collgen to observe the output ([Screenshot10](#)). Observing this output reveals that no extra zeros are appended to the file ([Screenshot11](#)) whereas when the same procedure is performed with a non-multiple of 64-byte length file ([Screenshot12](#)), appended zeros can be observed ([Screenshot13](#)). When the binary executables for each of these files are compared against each other ([Screenshot14](#))([Screenshot15](#)), the bytes that can be observed as different are in the decimal byte positions of 84, 123, 148, 174, 175, 110, and 188. In each instance, the hexadecimal value was altered by only one bit. For example, the value 0x16 was changed to 0x96 at byte 84 and the value 0x34 was changed to 0xB4 at byte 174. Further, the transformation of each byte was by exactly 128 in value with the exception of the transformation from 0x5A to 0x59 producing a change value of 1. This feature of MD5 collisions is very interesting as it reveals the types of changes that produce similar hash values, such as transformations of exactly 128 in value. However, despite these differences, comparing the hash values continues to show identical values highlighting a collision has occurred with the mapping between the hashed values and the inputs ([Screenshot16](#)).

The second task of the lab then introduced a fundamental property of MD5 hashes being that if the same value is appended to two files with the same hash value, they will persist to have matching hash values (Long, 2019). This can be verified by concatenating the same

value to the end of two values returned from md5collgen and comparing the hashed sums ([Screenshot17](#)). Doing so will then verify that this property is true and is further upheld when appended to either end of the file as long as it is appended to the same end on both files. This property is critical to collision attacks as it then allows for malicious code to be embedded within code that already hashes with the value to be achieved. As will be explored in the next steps, this vulnerability can then be exploited to modify code in both output and functionality.

Task three begins to explore this property by instructing for the generation of two input files that vary in data, but hash to the same value. To accomplish this task, a source code file was first created in C where I filled an array of length 200 with Xs ([Screenshot18](#)). The reason it was chosen to fill the array with exclusively one letter is so that the location of the array would be easier to spot in the binary executable file with many identical characters in sequence. After this, the source file was compiled into an executable ([Screenshot19](#)) and observed within the bless editor. Observing this file in bless, the revealed that the array started at decimal index 4113 ([Screenshot20](#)). The next step for this task was then to parse out the middle section of the file containing the array for modification with the md5collgen function before piecing the file back together through concatenation. For this reason, I would need to parse a 128-byte region from the section of the file containing the array, leaving the rest to be included in the prefix or suffix for the file. The prefix for the array needs to be a multiple of 64-bytes to ensure the file can be pieced back together without extra padding being added and for this reason, the start of the array at index 4113 was not a suitable start location. I then chose index 4160 as the starting index for this block and set the ending point 128 bytes later at index 4288. This was accomplished by first using the head and tail

functions to gain the prefix and suffix outside of these bounds ([Screenshot21](#)) and then using the bless editor to remove the prefix ([Screenshot22](#)) and suffix ([Screenshot23](#)) and save the remaining 128 bytes as a new file ([Screenshot24](#)). Both the newly created prefix and suffix were then run through the md5collgen function to retrieve new values for each ([Screenshot25](#)) and these values were compared using the cmp command to identify differing characters ([Screenshot26](#)). As an intermediary step, all of the newly generated prefixes and suffixes were checked using md5sum to confirm that the hash values for these elements persisted to match up ([Screenshot27](#)). Following this, the 128-byte array value was parsed out of each prefix using the tail command ([Screenshot28](#)) before the prefixes, suffixes, and array values were concatenated back together to produce the new files. The hash values for each of these values were then compared and shown to match ([Screenshot29](#)) demonstrating the concatenation property ([Screenshot30](#)) and completing task three.

The final task then required the implementation of a file that not only represented a change in data but in functionality. Through using concatenation within a file, the fourth task demonstrates how a malicious file could be made to hash with a non-malicious file using MD5 collision exploitation (Long, 2019). To demonstrate this concept, I began by creating a file with two arrays of length 200 filled with Xs ([Screenshot31](#)). This file also prints that the code is safe when all of the values in each array match and that the code is malicious when they do not ([Screenshot32](#)). Using bless to examine the executable, the long string of Xs is easily located ([Screenshot33](#)). The two arrays can then be distinguished by adding 200 to the starting X index of 4112 to see that the second array begins at index 4312. However, as the prefix will need to be of a length divisible by 64, the starting bit for the middle section to be parsed out was switched to 4160 with the 128-bytes ending at index 4288. As before, the

prefix and suffix could then be created through the use of the head and tail commands ([Screenshot34](#)) and made collision-ready ([Screenshot35](#)). The outputs from md5collgen could then be parsed to retrieve a value for both the “safe” and “malicious” code ([Screenshot36](#)) that hash to the same value ([Screenshot37](#)). With these extracted, the two files could then be built back together ([Screenshot38](#)) and as a concatenation is performed exclusively of elements with matching hash values, the final product maintains the matching hash values.

### **Lab Analysis:**

This lab highlighted many interesting features such as the effect of the processing block size, byte transformation value, and concatenation properties of MD5 hashes. Having an understanding of the elements that affect the final hash value allows for the clever modification of values in such a way that the final hash remains unaltered (Long, 2019). This is a serious concern as the loss of the ability to validate that a hash came from a single input undermines the ability to use the hashing algorithm. The loss of this property would then disable the ability to compare hashes as two like hashes may have come from different sources or enable malicious code to masquerade as benign code with the same hash sum. The high rate of collisions in MD5 highlights a serious vulnerability (Mohammed Ali & Kadhim Farhan, 2020), and from this, it is then easy for a security professional to understand the deprecation of this algorithm (Obaida et al., 2022). While the MD5 function is no longer in use, studying the function provides a greater understanding of hashing functions as a whole and the security implications of hashing collisions.

When thinking of a common context in which hashing is regularly used such as password storage or database lookups (Shen et al., 2022), it is clear to see the potential dangers of MD5



collision exploits. If a password value is hashed but has several other inputs that could hash to that same value, then the security of the password is greatly diminished. Likewise, when searching a database through stored hash values, the hash values could be exploited to index into unintended areas of the database. For this reason, security professionals working in an environment where hash functions are used must understand the implications of their chosen hash function and ensure that vulnerabilities of collisions are mitigated.

**Conclusion:**

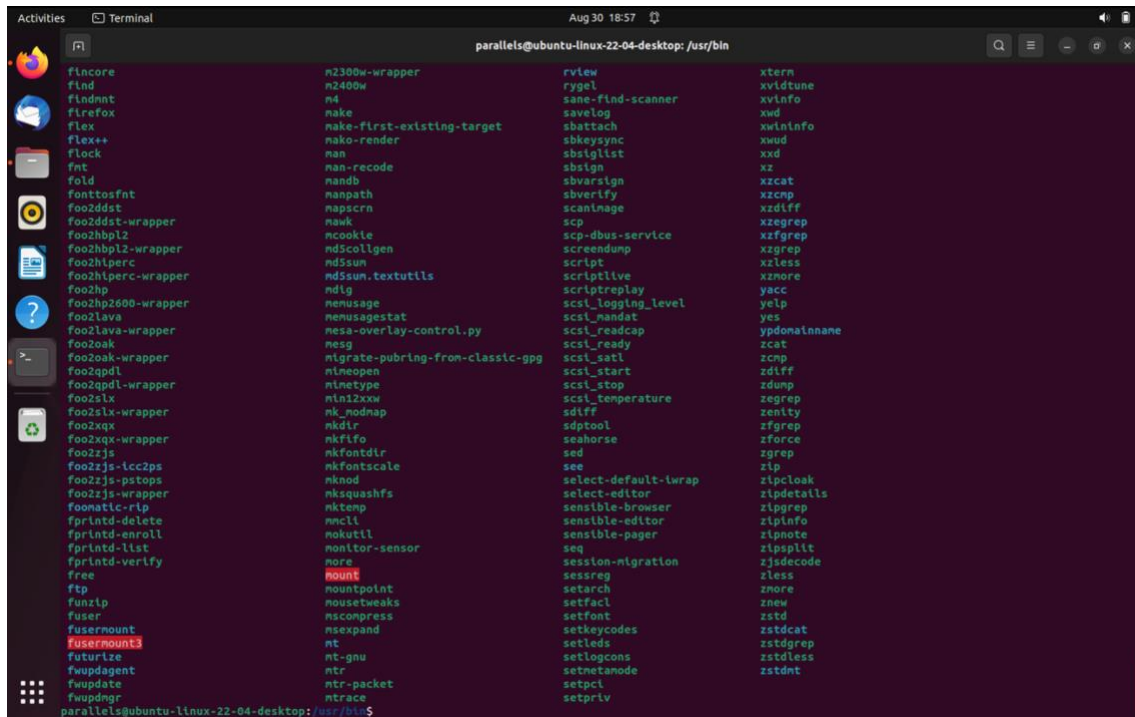
Despite the security concerns involved with deprecated hash functions, hashing remains an extremely relevant and useful form of modern cryptography (Microsoft, 2022). Understanding the vulnerabilities associated with collision attacks then allows a security professional to implement hash functions securely and address potential vulnerabilities before they are exploited. Further, non-deprecated algorithms could be chosen or modified to increase security and mitigate the risk of collisions (Alenezi et al., 2022). Growth in this knowledge can then be gained through researching and exploring deprecated algorithms such as MD5 or SHA1 and understanding the reasons why they are vulnerable to attack. Seeking to understand these algorithms can then promote the principles of integrity and non-repudiation through the secure application of hashing algorithms.

## References

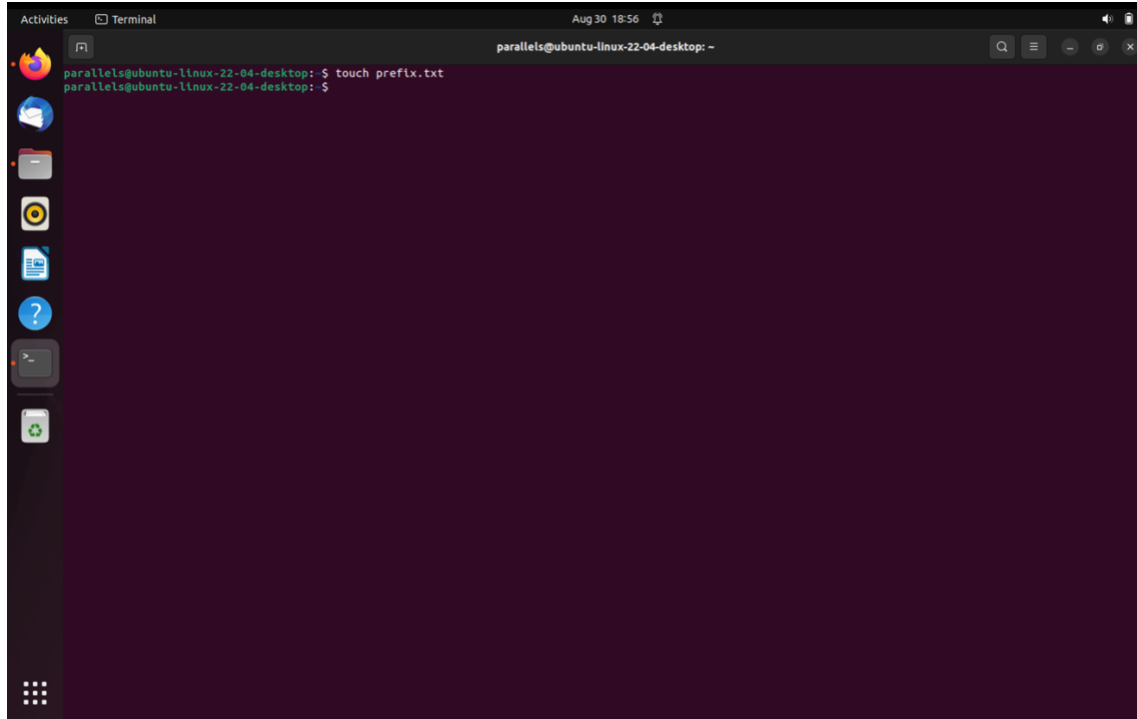
- Alenezi, M. N., Alabdulrazzaq, H., & Mohammad, N. Q. (2020). Symmetric encryption algorithms: Review and evaluation study. *International Journal of Communication Networks and Information Security*, 12(2), 256-272.  
<https://www.proquest.com/docview/2440677681?pq-origsite=summon&accountid=12085>
- Amazon. (2023). *AWS cryptography services: AWS cryptographic services and tools guide*. Amazon. <https://docs.aws.amazon.com/pdfs/crypto/latest/userguide/crypto-ug.pdf#concepts-algorithms>
- Basta, A. (2018). *Oriyano, cryptography: Infosec pro guide*. McGraw-Hill Education.  
<https://bookshelf.vitalsource.com/reader/books/9781307297003/pageid/14>
- Microsoft. (March 11, 2022). *Overview of encryption, digital signatures, and hash algorithms in .NET*. Microsoft. <https://learn.microsoft.com/en-us/dotnet/standard/security/cryptographic-services>
- Long, S. (2019). A comparative analysis of the application of hashing encryption algorithms for MD5, SHA-1, and SHA-512. *Journal of Physics. Conference Series*, 1314(1), 12210. <https://doi.org/10.1088/1742-6596/1314/1/012210>
- Mohammed Ali, A., & Kadhim Farhan, A. (2020). A novel improvement with an effective expansion to enhance the MD5 hash function for verification of a secure E-document. *IEEE Access*, 8, 80290-80304. <https://doi.org/10.1109/ACCESS.2020.2989050>
- Obaida, T. H., Salman, H. A., & Zugair, H. N. (2022). Improve MD5 hash function for document authentication. *Webology*, 19(1), 7223-7234.

<https://www.proquest.com/docview/2692792776?accountid=12085&forcedol=true&forcedol=true&pq-origsite=summon>

Shen, Y., Wu, T., Wang, G., Dong, X., & Qian, H. (2022). Improved collision detection of MD5 using sufficient condition combination. The British Computer Society. *Computer Journal*. <https://doi.org/10.1093/comjnl/bxab109>



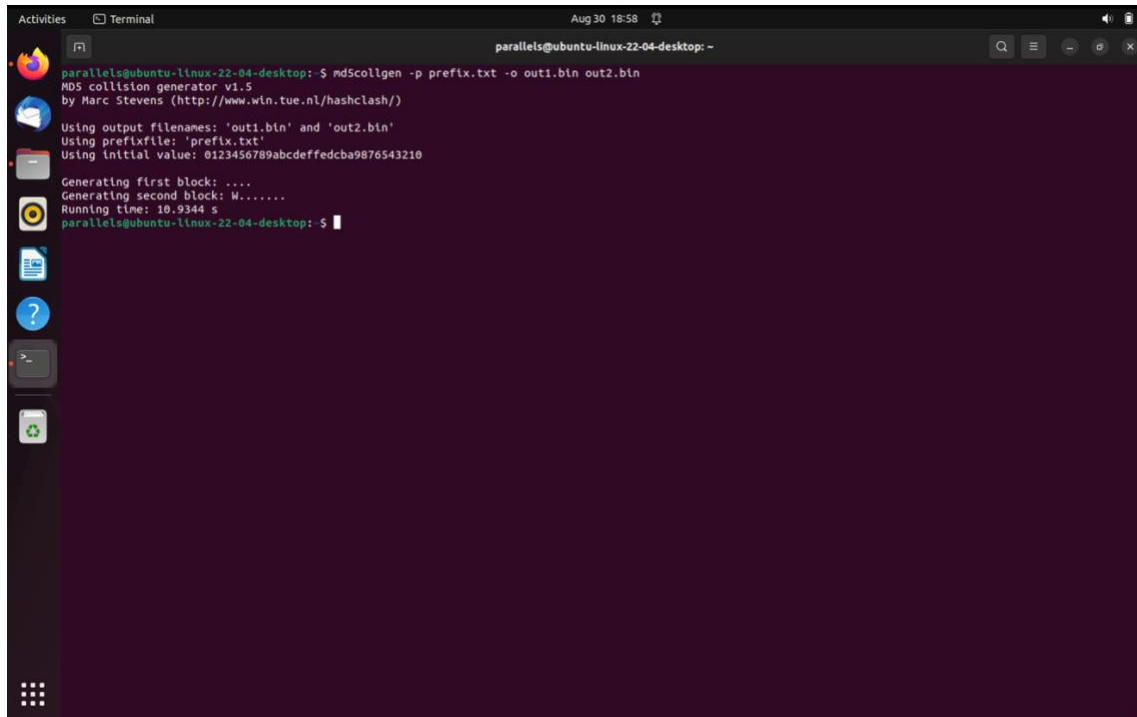
Screenshot3: ([Return to text](#))



A terminal window titled "Terminal" with a dark background and light text. The window shows the command prompt "parallels@ubuntu-linux-22-04-desktop: ~" and the command "touch prefix.txt" being executed. The output is "parallels@ubuntu-linux-22-04-desktop: \$". The window has a title bar with "Activities" and "Terminal" tabs, and a status bar at the bottom showing "Aug 30 18:56".

```
parallels@ubuntu-linux-22-04-desktop: ~  
parallels@ubuntu-linux-22-04-desktop: $ touch prefix.txt  
parallels@ubuntu-linux-22-04-desktop: $
```

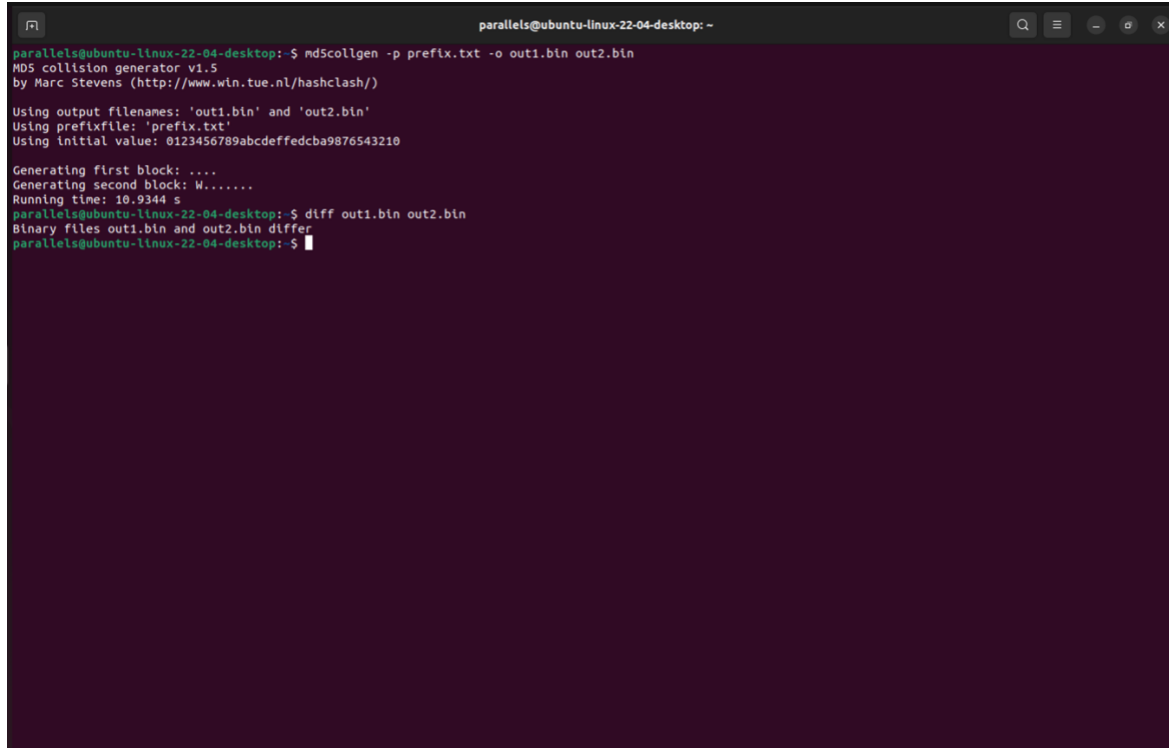
Screenshot4: ([Return to text](#))



A terminal window titled "Terminal" with a dark background and light text. The window shows the command prompt "parallels@ubuntu-linux-22-04-desktop: ~" and the command "md5collgen -p prefix.txt -o out1.bin out2.bin" being executed. The output is "MD5 collision generator v1.5 by Marc Stevens (http://www.win.tue.nl/hashclash/)", "Using output filenames: 'out1.bin' and 'out2.bin'", "Using prefixfile: 'prefix.txt'", "Using initial value: 0123456789abcdefdcba9876543210", "Generating first block: ....", "Generating second block: W.....", "Running time: 10.9344 s", and "parallels@ubuntu-linux-22-04-desktop: \$". The window has a title bar with "Activities" and "Terminal" tabs, and a status bar at the bottom showing "Aug 30 18:58".

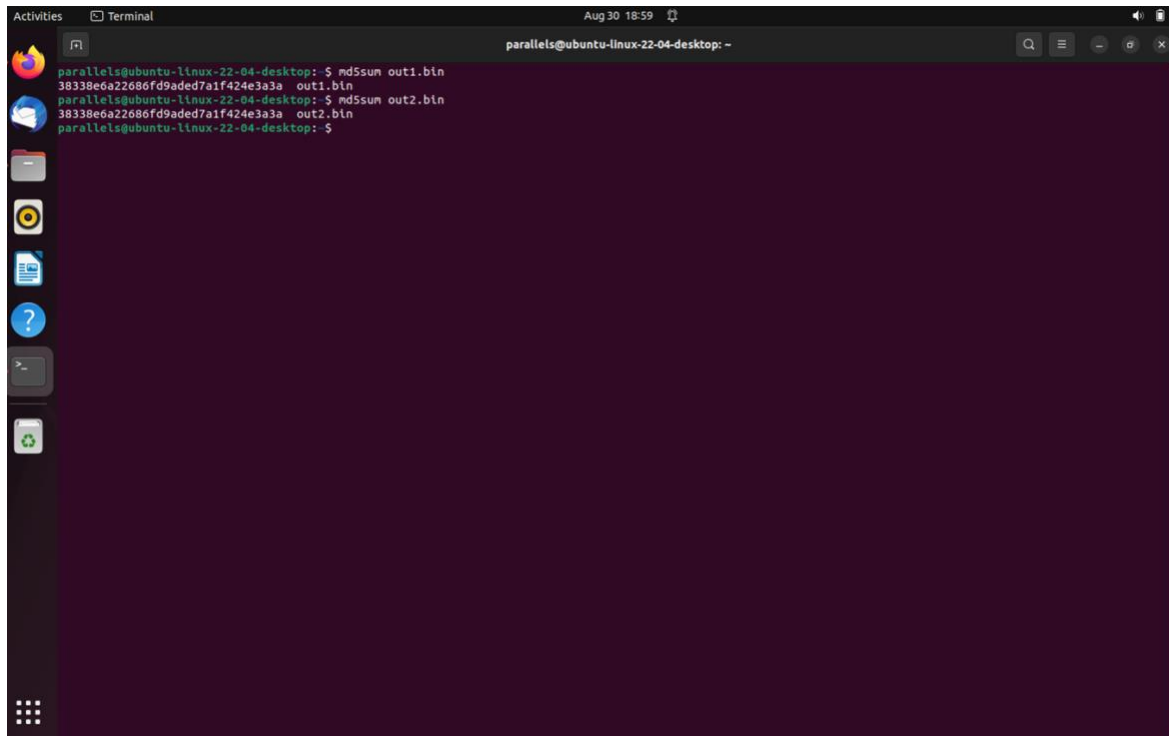
```
parallels@ubuntu-linux-22-04-desktop: ~  
parallels@ubuntu-linux-22-04-desktop: $ md5collgen -p prefix.txt -o out1.bin out2.bin  
MD5 collision generator v1.5  
by Marc Stevens (http://www.win.tue.nl/hashclash/)  
Using output filenames: 'out1.bin' and 'out2.bin'  
Using prefixfile: 'prefix.txt'  
Using initial value: 0123456789abcdefdcba9876543210  
Generating first block: ....  
Generating second block: W.....  
Running time: 10.9344 s  
parallels@ubuntu-linux-22-04-desktop: $
```

Screenshot5: ([Return to text](#))

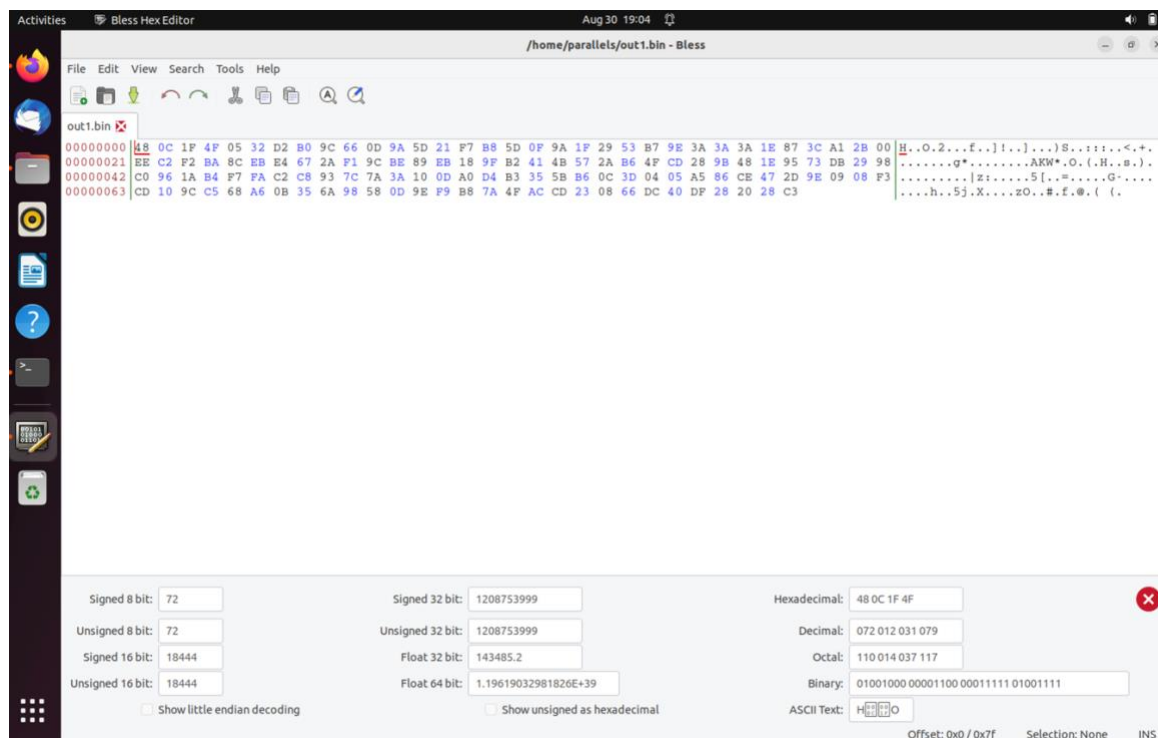
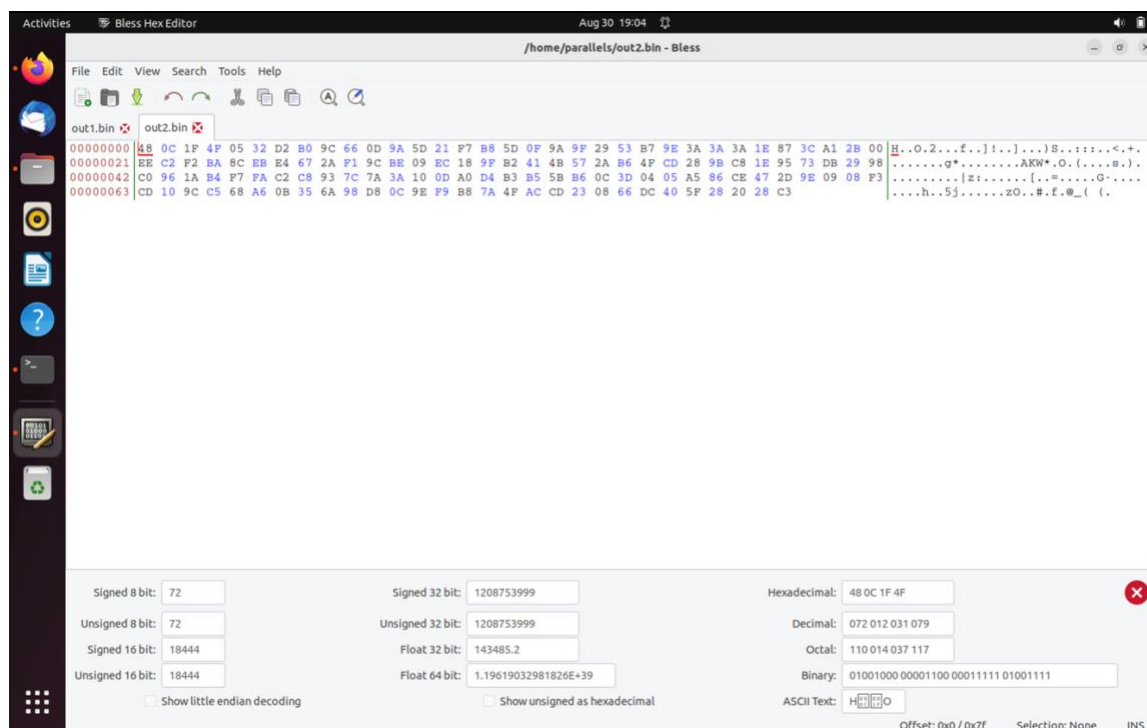
A terminal window titled 'parallels@ubuntu-linux-22-04-desktop: ~' with a search icon and window controls. The terminal shows the execution of 'md5collgen -p prefix.txt -o out1.bin out2.bin'. It displays version information (v1.5 by Marc Stevens), output filenames, prefix file, and initial value. It then shows the generation of two blocks and a diff command result indicating the files differ.

```
parallels@ubuntu-linux-22-04-desktop: ~  
parallels@ubuntu-linux-22-04-desktop:~$ md5collgen -p prefix.txt -o out1.bin out2.bin  
MD5 collision generator v1.5  
by Marc Stevens (http://www.win.tue.nl/hashclash/)  
  
Using output filenames: 'out1.bin' and 'out2.bin'  
Using prefixfile: 'prefix.txt'  
Using initial value: 0123456789abcdeffedcba9876543210  
  
Generating first block: ....  
Generating second block: W.....  
Running time: 10.9344 s  
parallels@ubuntu-linux-22-04-desktop:~$ diff out1.bin out2.bin  
Binary files out1.bin and out2.bin differ  
parallels@ubuntu-linux-22-04-desktop:~$
```

Screenshot6: ([Return to text](#))

A terminal window titled 'parallels@ubuntu-linux-22-04-desktop: ~' with a search icon and window controls. The terminal shows the execution of 'md5sum out1.bin' and 'md5sum out2.bin', both resulting in the same hash: 38338e6a22686fd9aded7a1f424e3a3a.

```
parallels@ubuntu-linux-22-04-desktop: ~  
parallels@ubuntu-linux-22-04-desktop:~$ md5sum out1.bin  
38338e6a22686fd9aded7a1f424e3a3a  out1.bin  
parallels@ubuntu-linux-22-04-desktop:~$ md5sum out2.bin  
38338e6a22686fd9aded7a1f424e3a3a  out2.bin  
parallels@ubuntu-linux-22-04-desktop:~$
```

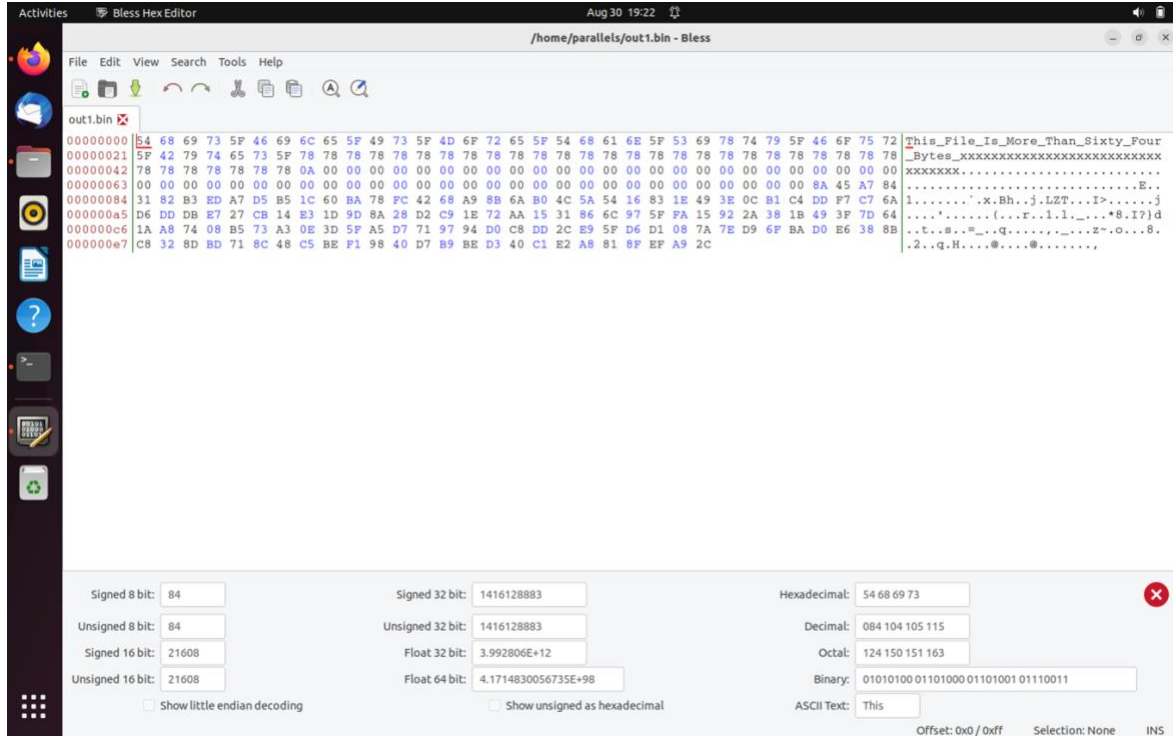
Screenshot7: ([Return to text](#))Screenshot8: ([Return to text](#))



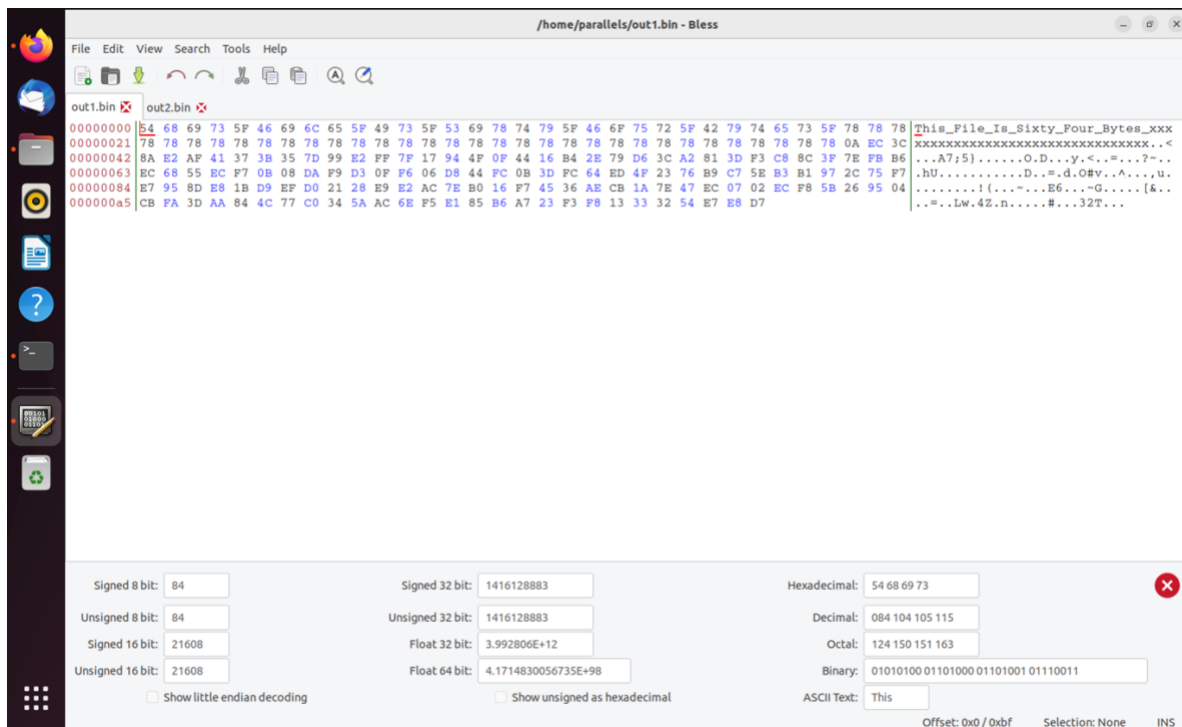


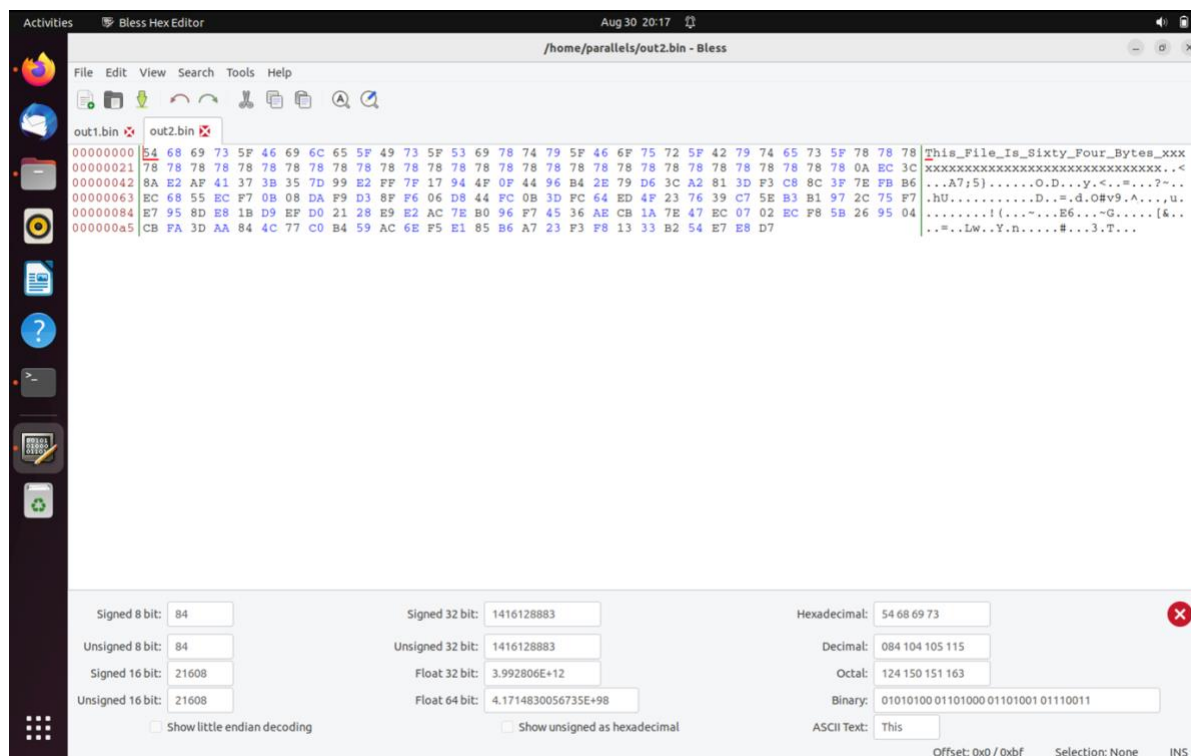
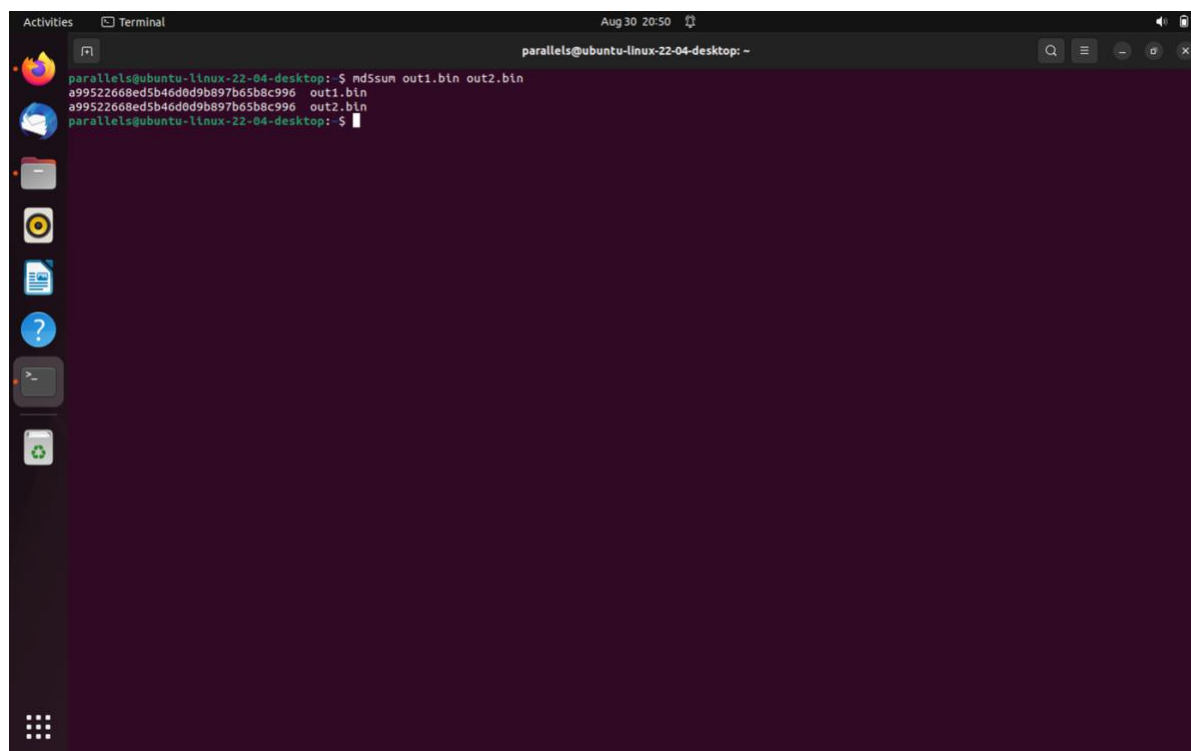


Screenshot13: ([Return to text](#))



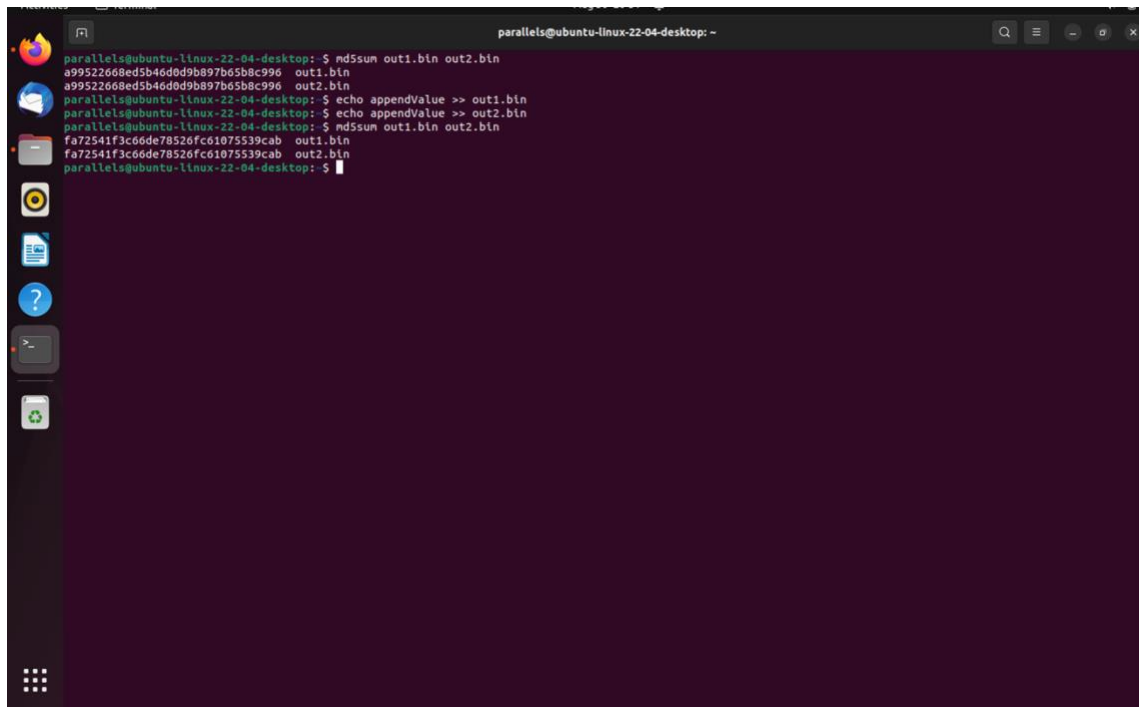
Screenshot14: ([Return to text](#))



Screenshot15: ([Return to text](#))Screenshot16: ([Return to text](#))

## Screenshots: Task 2

Screenshot17: ([Return to text](#))



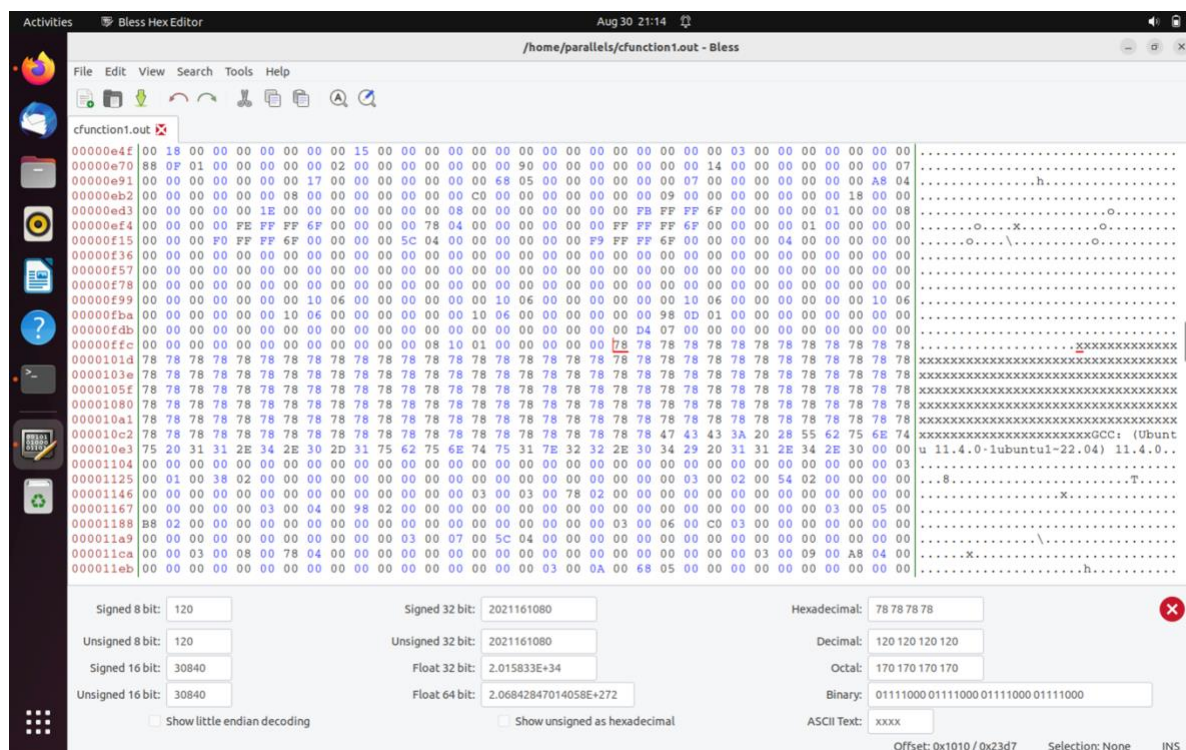
The screenshot shows a terminal window titled "parallels@ubuntu-linux-22-04-desktop: ~". The terminal displays the following commands and their outputs:

```
parallels@ubuntu-linux-22-04-desktop: ~$ md5sum out1.bin out2.bin
a99522668ed5b46d0d9b897b05b8c996  out1.bin
a99522668ed5b46d0d9b897b05b8c996  out2.bin
parallels@ubuntu-linux-22-04-desktop: ~$ echo appendValue >> out1.bin
parallels@ubuntu-linux-22-04-desktop: ~$ echo appendValue >> out2.bin
parallels@ubuntu-linux-22-04-desktop: ~$ md5sum out1.bin out2.bin
fa72541f3c66de78526fc61075539cab  out1.bin
fa72541f3c66de78526fc61075539cab  out2.bin
parallels@ubuntu-linux-22-04-desktop: ~$
```

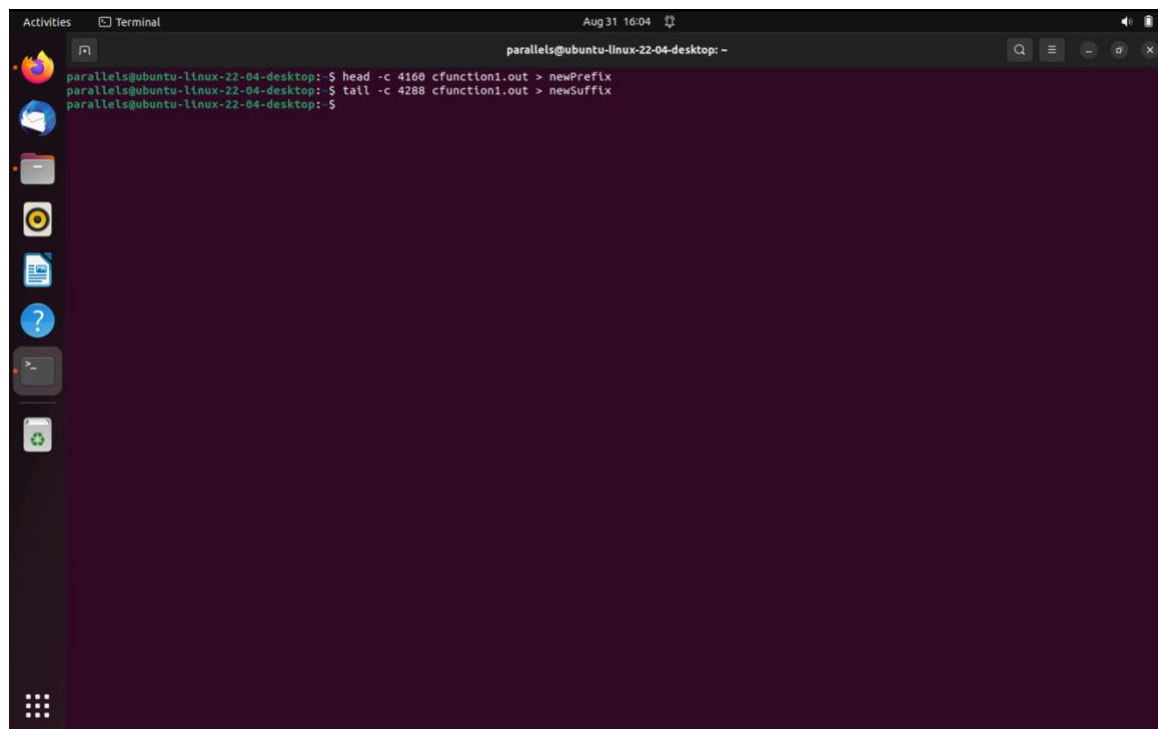
The terminal window has a dark purple background and a light gray border. The left sidebar shows various application icons, including a file manager, a web browser, and a terminal. The top of the window has a title bar with standard window controls (minimize, maximize, close) and a search icon.



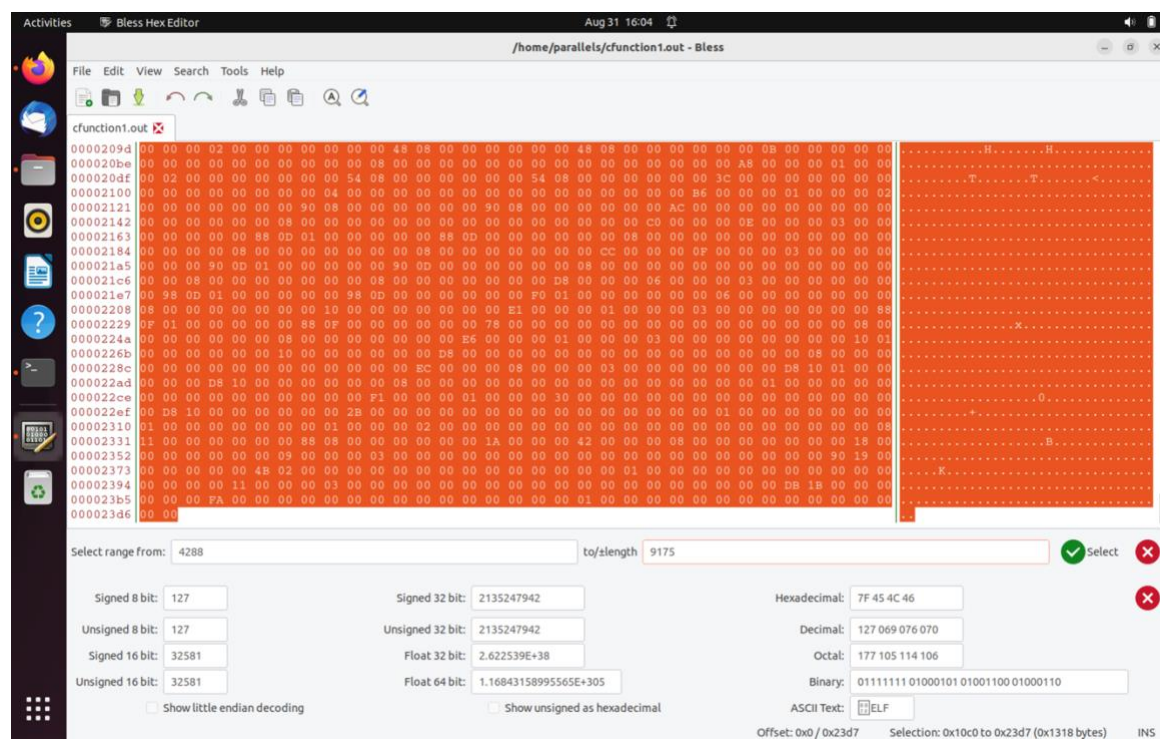
Screenshot20: ([Return to text](#))

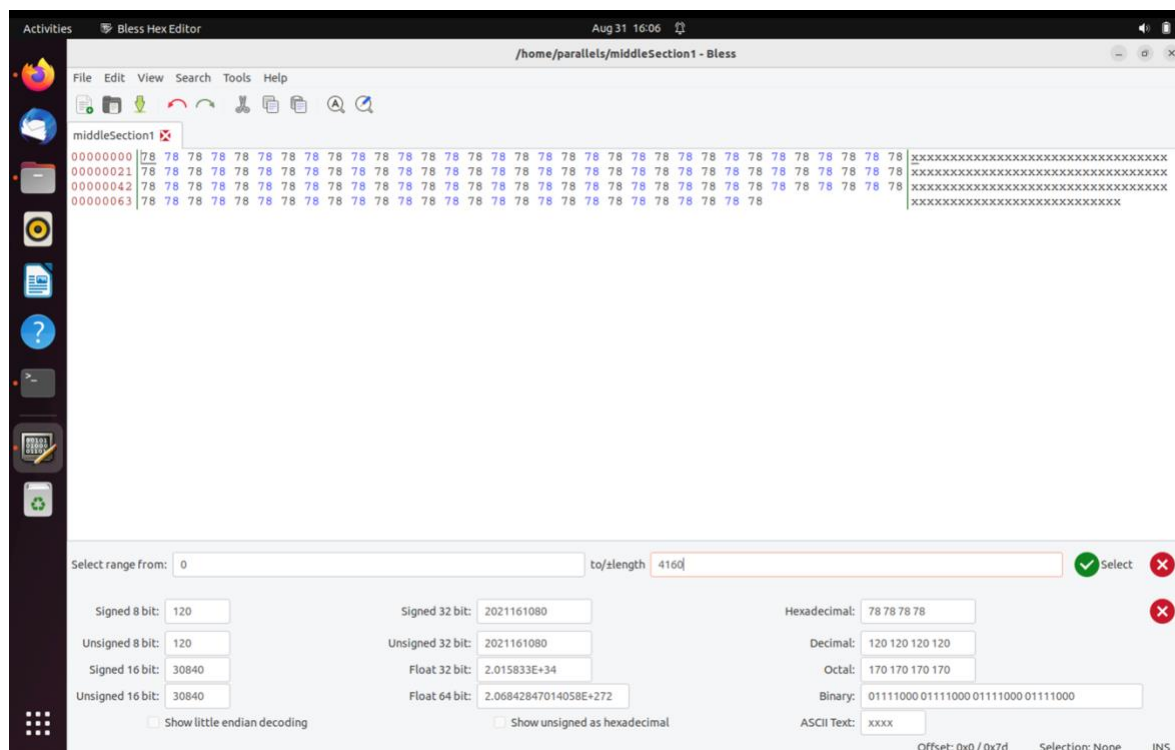
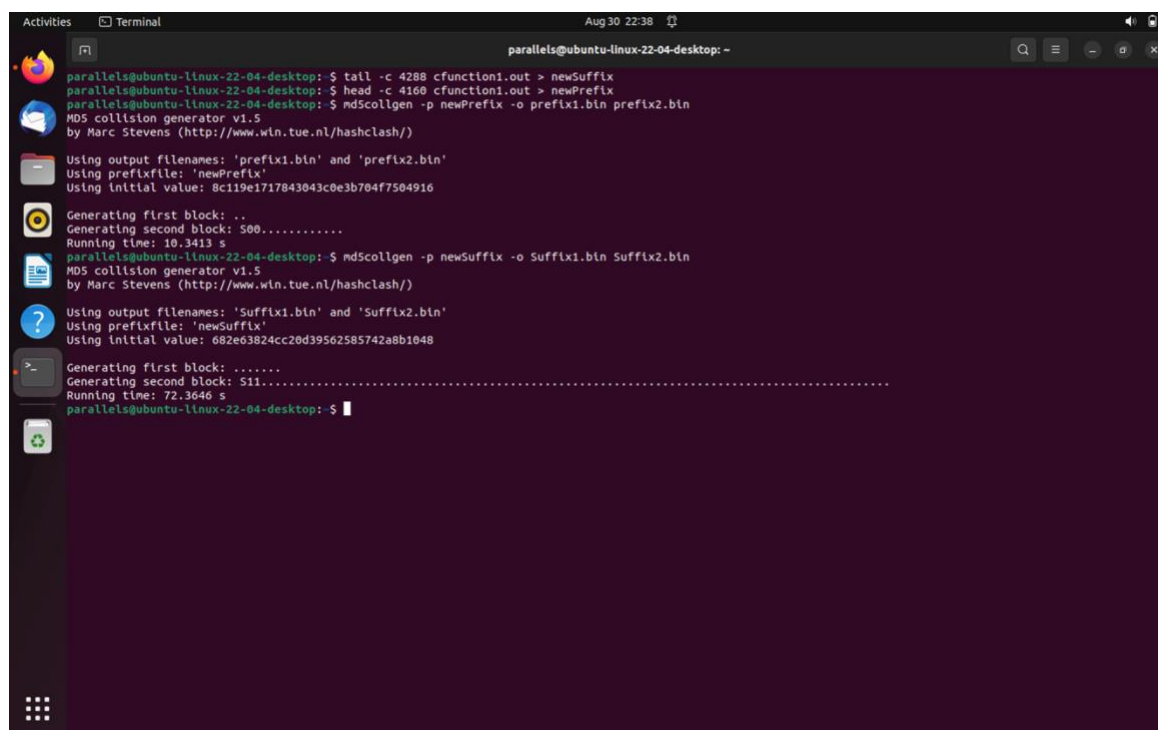


Screenshot21: ([Return to text](#))







Screenshot24: ([Return to text](#))Screenshot25: ([Return to text](#))



Screenshot26: ([Return to text](#))

```

parallel@ubuntu-linux-22-04-desktop: $ tail -c 4288 cfunction1.out > newSuffix
parallel@ubuntu-linux-22-04-desktop: $ head -c 4160 cfunction1.out > newPrefix
parallel@ubuntu-linux-22-04-desktop: $ md5collgen -p newPrefix -o prefix1.bin prefix2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'prefix1.bin' and 'prefix2.bin'
Using prefixfile: 'newPrefix'
Using initial value: 8c119e1717843043c0e3b704f7504916

Generating first block: ..
Generating second block: 500.....
Running time: 10.3413 s
parallel@ubuntu-linux-22-04-desktop: $ md5collgen -p newSuffix -o Suffix1.bin Suffix2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'Suffix1.bin' and 'Suffix2.bin'
Using prefixfile: 'newSuffix'
Using initial value: 682e63824cc20d39562585742a8b1048

Generating first block: .....
Generating second block: 511.....
Running time: 72.3646 s
parallel@ubuntu-linux-22-04-desktop: $ cmp -lb prefix?.bin
4180 340 M-: 140 ^
4206 270 M-8 70 8
4207 25 ^U 26 ^V
4220 272 M-: 72 :
4244 37 ^_ 237 M-^_
4270 55 - 255 M--
4271 107 G 106 F
4284 77 ? 277 M-?
parallel@ubuntu-linux-22-04-desktop: $

```

Screenshot27: ([Return to text](#))

```

parallel@ubuntu-linux-22-04-desktop: $ md5sum prefix1.bin
c89c360136b675d05e22c3ba8f41fda8 prefix1.bin
parallel@ubuntu-linux-22-04-desktop: $ md5sum prefix2.bin
c89c360136b675d05e22c3ba8f41fda8 prefix2.bin
parallel@ubuntu-linux-22-04-desktop: $ md5sum Suffix1.bin
9ae2a14236aefcf65d463a09c51e696b Suffix1.bin
parallel@ubuntu-linux-22-04-desktop: $ md5sum Suffix2.bin
9ae2a14236aefcf65d463a09c51e696b Suffix2.bin
parallel@ubuntu-linux-22-04-desktop: $

```

Screenshot28: ([Return to text](#))

```

parallel@ubuntu-linux-22-04-desktop:~$ head -c 4224 cfunction1.out > newPrefix
parallel@ubuntu-linux-22-04-desktop:~$ md5collgen -p newPrefix -o prefix1.bin prefix2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'prefix1.bin' and 'prefix2.bin'
Using prefixfile: 'newPrefix'
Using initial value: 90f9e2b06ba4981a5e2f6f4d476fc2f7

Generating first block: .....
Generating second block: W.....
Running time: 28.8259 s
parallel@ubuntu-linux-22-04-desktop:~$ tail -c +4352 cfunction1.out > newSuffix
parallel@ubuntu-linux-22-04-desktop:~$ md5collgen -p newSuffix -o Suffix1.bin Suffix2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'Suffix1.bin' and 'Suffix2.bin'
Using prefixfile: 'newSuffix'
Using initial value: 76f3cc096ee6c80ee907767e0f248341

Generating first block: ..
Generating second block: S10.....
Running time: 4.72501 s
parallel@ubuntu-linux-22-04-desktop:~$ tail -c 128 prefix1.bin > middleSection1
parallel@ubuntu-linux-22-04-desktop:~$ tail -c 128 prefix2.bin > middleSection2
parallel@ubuntu-linux-22-04-desktop:~$ cat newPrefix middleSection1 newSuffix > firstCFFunction
parallel@ubuntu-linux-22-04-desktop:~$ cat newPrefix middleSection2 newSuffix > secondCFFunction
parallel@ubuntu-linux-22-04-desktop:~$ md5sum firstCFFunction
1cb1707d7252041dc6ba0eba22d60323 firstCFFunction
parallel@ubuntu-linux-22-04-desktop:~$ md5sum secondCFFunction
1cb1707d7252041dc6ba0eba22d60323 secondCFFunction
parallel@ubuntu-linux-22-04-desktop:~$

```

Screenshot29: ([Return to text](#))

```

parallel@ubuntu-linux-22-04-desktop:~$ head -c 4224 cfunction1.out > newPrefix
parallel@ubuntu-linux-22-04-desktop:~$ md5collgen -p newPrefix -o prefix1.bin prefix2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'prefix1.bin' and 'prefix2.bin'
Using prefixfile: 'newPrefix'
Using initial value: 90f9e2b06ba4981a5e2f6f4d476fc2f7

Generating first block: .....
Generating second block: W.....
Running time: 28.8259 s
parallel@ubuntu-linux-22-04-desktop:~$ tail -c +4352 cfunction1.out > newSuffix
parallel@ubuntu-linux-22-04-desktop:~$ md5collgen -p newSuffix -o Suffix1.bin Suffix2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'Suffix1.bin' and 'Suffix2.bin'
Using prefixfile: 'newSuffix'
Using initial value: 76f3cc096ee6c80ee907767e0f248341

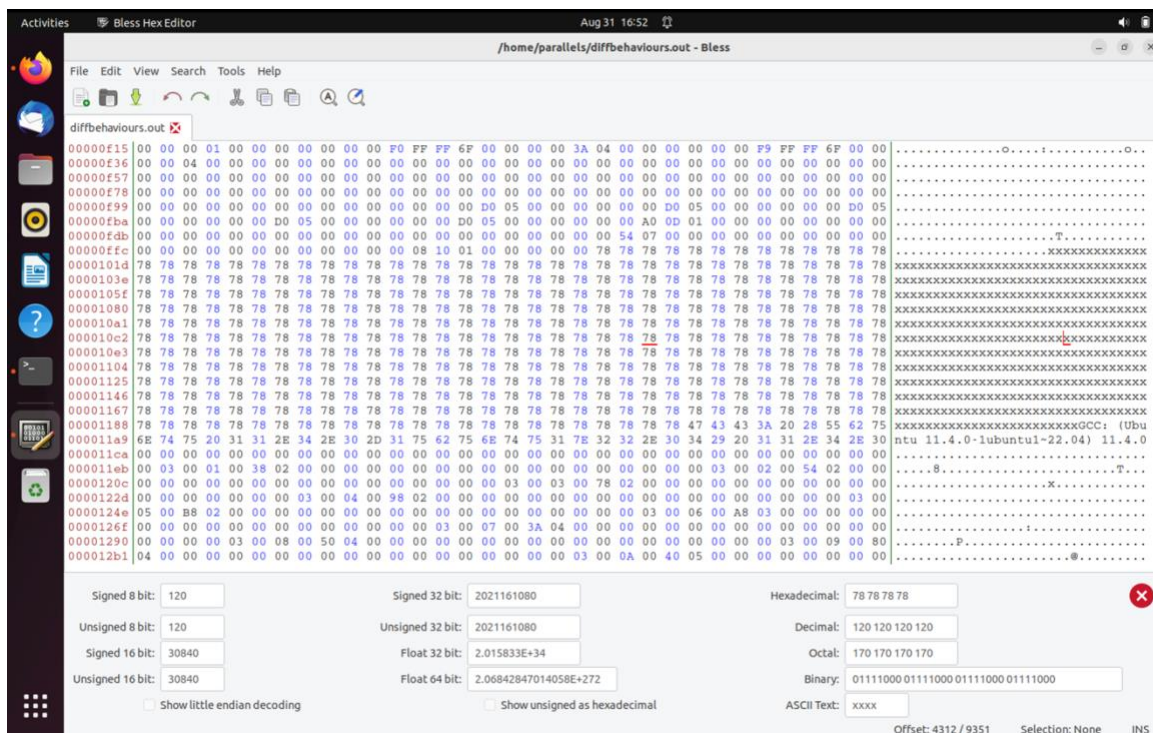
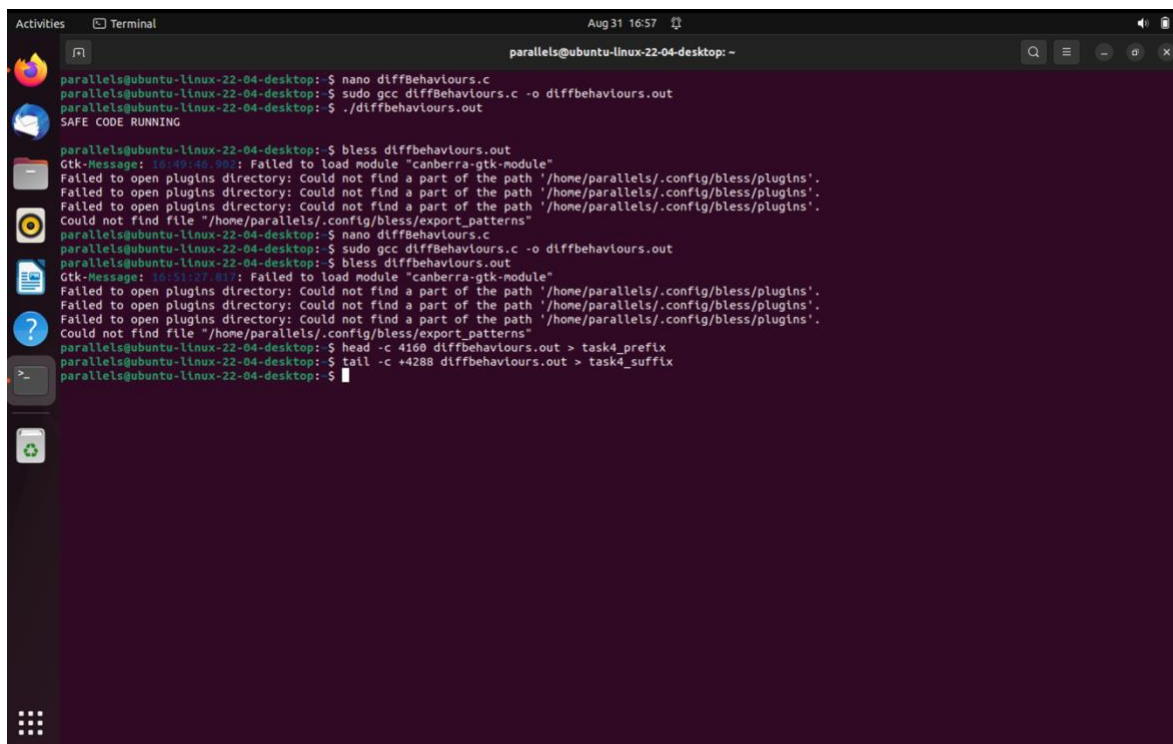
Generating first block: ..
Generating second block: S10.....
Running time: 4.72501 s
parallel@ubuntu-linux-22-04-desktop:~$ tail -c 128 prefix1.bin > middleSection1
parallel@ubuntu-linux-22-04-desktop:~$ tail -c 128 prefix2.bin > middleSection2
parallel@ubuntu-linux-22-04-desktop:~$ cat newPrefix middleSection1 newSuffix > firstCFFunction
parallel@ubuntu-linux-22-04-desktop:~$ cat newPrefix middleSection2 newSuffix > secondCFFunction
parallel@ubuntu-linux-22-04-desktop:~$ md5sum firstCFFunction
1cb1707d7252041dc6ba0eba22d60323 firstCFFunction
parallel@ubuntu-linux-22-04-desktop:~$ md5sum secondCFFunction
1cb1707d7252041dc6ba0eba22d60323 secondCFFunction
parallel@ubuntu-linux-22-04-desktop:~$ cmp -lb firstCFFunction secondCFFunction
cmp: firstCFFunction: No such file or directory
parallel@ubuntu-linux-22-04-desktop:~$ cmp -lb firstCFFunction secondCFFunction
4244 76 > 276 M->
4270 375 M-) 175 )
4271 347 M-g 350 M-h
4284 220 M.-p 20 ^p
4308 47 ' 247 M-'
4334 170 x 370 M-x
4335 122 R 121 Q
4348 357 M-o 157 o
parallel@ubuntu-linux-22-04-desktop:~$

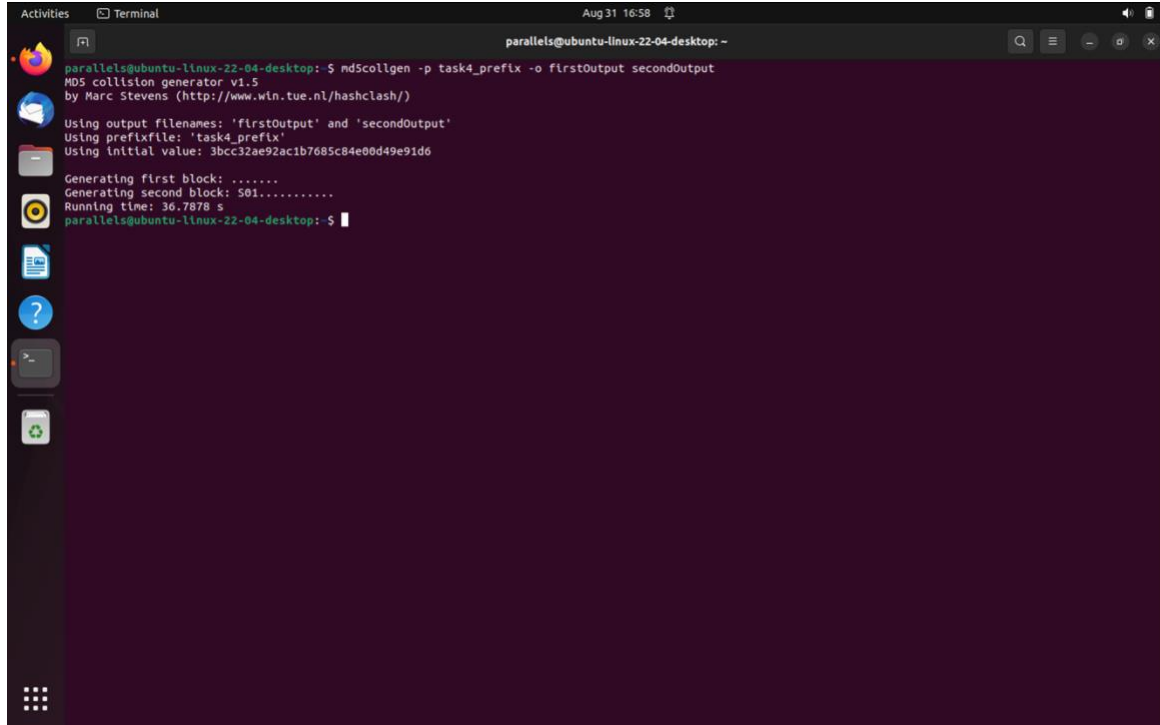
```







Screenshot33: ([Return to text](#))Screenshot34: ([Return to text](#))

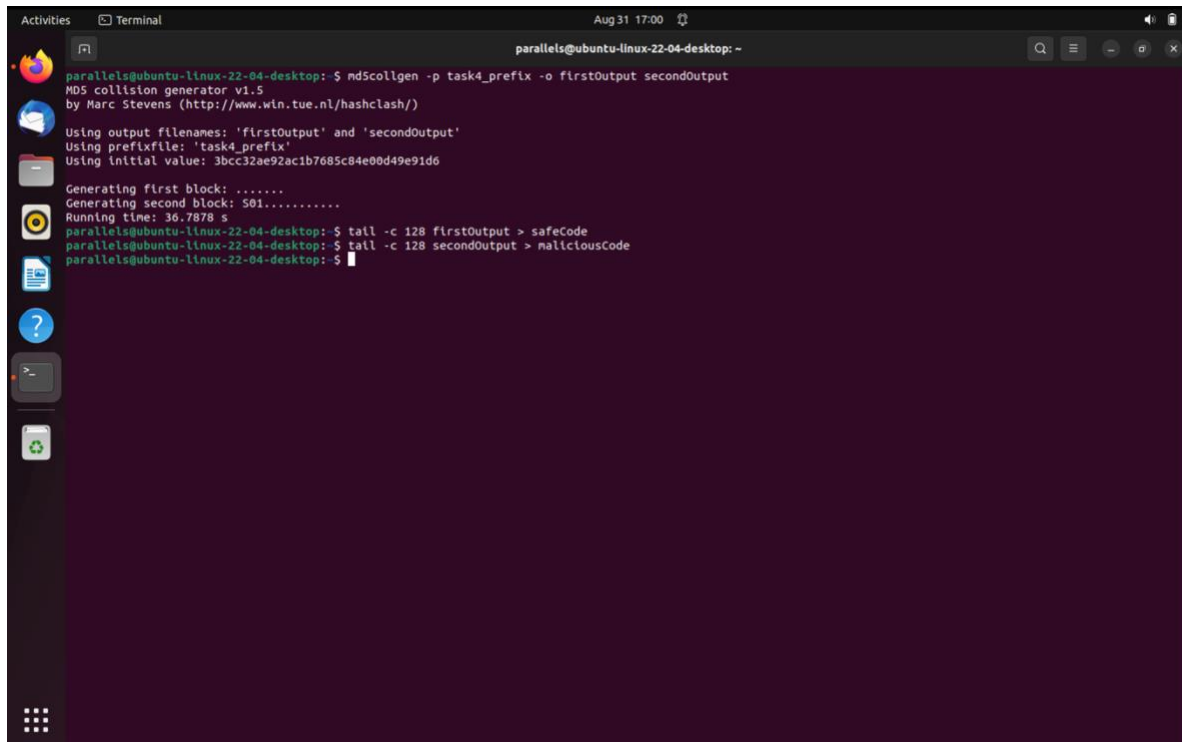
Screenshot35: ([Return to text](#))

A terminal window titled "Terminal" with a dark background and light text. The window shows the execution of the command `md5collgen -p task4_prefix -o firstOutput secondOutput`. The output indicates that the program is using output filenames 'firstOutput' and 'secondOutput', a prefix file 'task4\_prefix', and an initial value '3bcc32ae92ac1b7685c84e00d49e91d6'. It then shows the generation of the first and second blocks, with the second block being '501'. The running time is 36.7878 s. The prompt `parallels@ubuntu-linux-22-04-desktop:~$` is visible at the bottom.

```
parallels@ubuntu-linux-22-04-desktop:~$ md5collgen -p task4_prefix -o firstOutput secondOutput
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'firstOutput' and 'secondOutput'
Using prefixfile: 'task4_prefix'
Using initial value: 3bcc32ae92ac1b7685c84e00d49e91d6

Generating first block: .....
Generating second block: 501.....
Running time: 36.7878 s
parallels@ubuntu-linux-22-04-desktop:~$
```

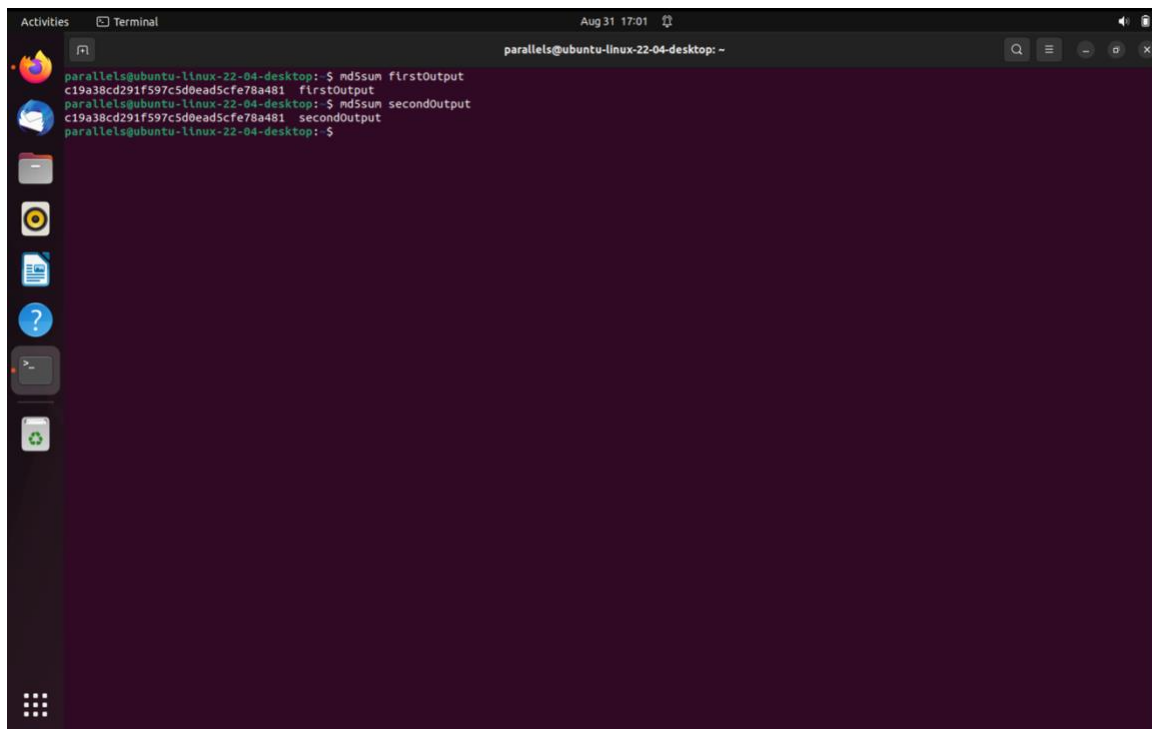
Screenshot36: ([Return to text](#))

A terminal window titled "Terminal" with a dark background and light text. The window shows the execution of the command `md5collgen -p task4_prefix -o firstOutput secondOutput`. The output indicates that the program is using output filenames 'firstOutput' and 'secondOutput', a prefix file 'task4\_prefix', and an initial value '3bcc32ae92ac1b7685c84e00d49e91d6'. It then shows the generation of the first and second blocks, with the second block being '501'. The running time is 36.7878 s. The prompt `parallels@ubuntu-linux-22-04-desktop:~$` is visible at the bottom. Below the prompt, the user enters `tail -c 128 firstOutput > safeCode` and `tail -c 128 secondOutput > maliciousCode`.

```
parallels@ubuntu-linux-22-04-desktop:~$ md5collgen -p task4_prefix -o firstOutput secondOutput
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'firstOutput' and 'secondOutput'
Using prefixfile: 'task4_prefix'
Using initial value: 3bcc32ae92ac1b7685c84e00d49e91d6

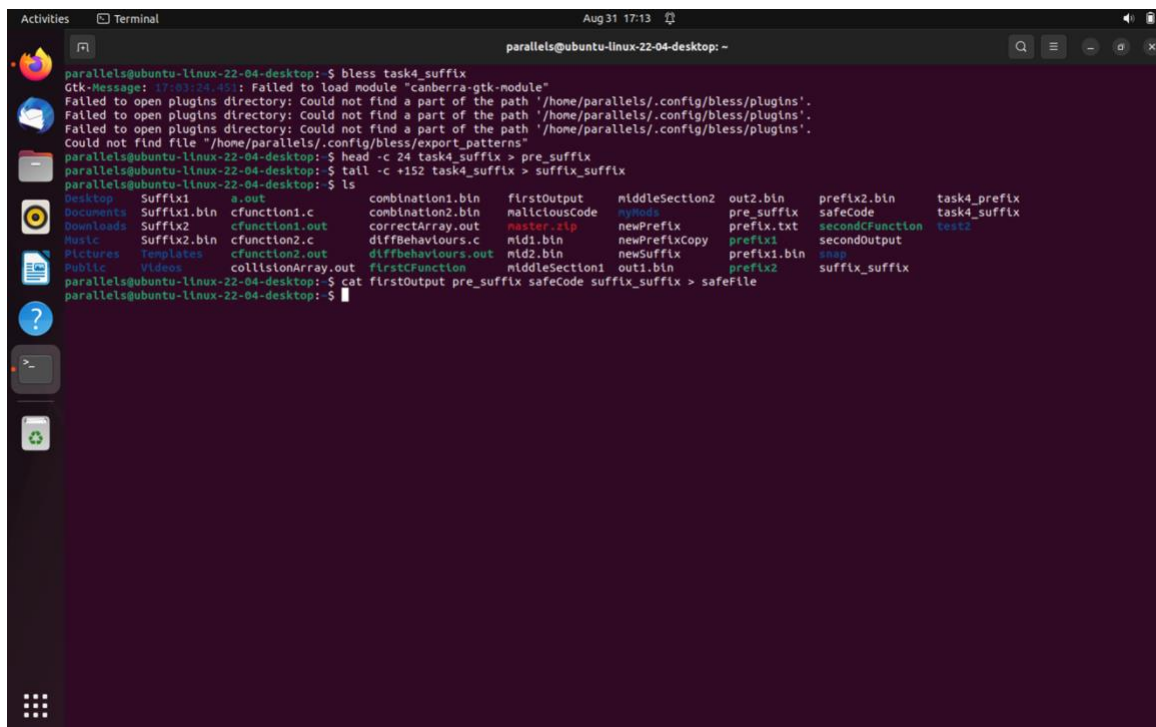
Generating first block: .....
Generating second block: 501.....
Running time: 36.7878 s
parallels@ubuntu-linux-22-04-desktop:~$ tail -c 128 firstOutput > safeCode
parallels@ubuntu-linux-22-04-desktop:~$ tail -c 128 secondOutput > maliciousCode
parallels@ubuntu-linux-22-04-desktop:~$
```

Screenshot37: ([Return to text](#))


```

parallel@ubuntu-linux-22-04-desktop:~$ md5sum firstOutput
c19a38cd291f597c5d0ead5cfe78a481  firstOutput
parallel@ubuntu-linux-22-04-desktop:~$ md5sum secondOutput
c19a38cd291f597c5d0ead5cfe78a481  secondOutput
parallel@ubuntu-linux-22-04-desktop:~$

```

Screenshot38: ([Return to text](#))


```

parallel@ubuntu-linux-22-04-desktop:~$ bless task4_suffix
Gtk-Message: 17:03:24.451: Failed to load module "canberra-gtk-module"
Failed to open plugins directory: Could not find a part of the path '/home/parallels/.config/bleess/plugins'.
Failed to open plugins directory: Could not find a part of the path '/home/parallels/.config/bleess/plugins'.
Failed to open plugins directory: Could not find a part of the path '/home/parallels/.config/bleess/plugins'.
Could not find file "/home/parallels/.config/bleess/export_patterns"
parallel@ubuntu-linux-22-04-desktop:~$ head -c 24 task4_suffix > pre_suffix
parallel@ubuntu-linux-22-04-desktop:~$ tail -c +152 task4_suffix > suffix_suffix
parallel@ubuntu-linux-22-04-desktop:~$ ls
Desktop  Suffix1  a.out      combination1.bin  firstOutput  middleSection2  out2.bin  prefix2.bin  task4_prefix
Documents Suffix1.bin cfunction1.c  combination2.bin  maliciousCode  none           pre_suffix  safeCode     task4_suffix
Downloads Suffix2.bin cfunction1.out correctArray.out  master.zip     newPrefix      prePrefix.txt  secondFunction test2
Music     Suffix2.bin cfunction2.c  diffBehaviours.c  mtd1.bin      newPrefixCopy  prefix1.txt   secondOutput
Pictures  Templates  cfunction2.out diffBehaviours.out mtd2.bin      newSuffix      prefix1.bin   snap
Public    Videos    collisionArray.out firstFunction      middleSection1 out1.bin       prefix2       suffix_suffix
parallel@ubuntu-linux-22-04-desktop:~$ cat firstOutput pre_suffix safeCode suffix_suffix > safeFile
parallel@ubuntu-linux-22-04-desktop:~$

```