Integrity Protection

Andrew McMullin
`mcmullin@umich.edu`

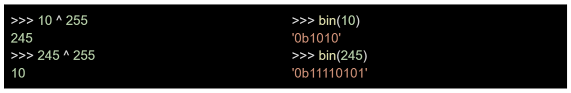
Abstract

Analyzing the effects of flipping bits to the MySQL encrypted '.ibd' data file, and the capabilities of some common encryption schemes on a basic '.csv' file. Started by flipping bits of a basic .csv file to see what effects flipping bits can have. We then look further at the effects of flipping bits on an encrypted .csv file to check for successful reloads, with incorrect data. Attempted to flip individual bits of the encrypted .ibd file to find any vulnerabilities in the Integrity Protection.

1 Introduction

My Project is to understand and expose vulnerabilities when using basic encryption techniques and attempt to make even the smallest of changes to the data without the public or private keys to manipulate the database. This will be attempted in the most common database management system MySQL and using a simple .csv file. A simple .csv file was added to this project in order to be a baseline and show effects in a simpler/understandable way. MySQL was chosen based on its popularity and its ability to be run locally on a local machine. With my limited understanding of the topic, I will first be attempting to find an Integrity Protection attack (which assures information has not been altered) because it is often neglected.

Integrity Protection is very important to how secure a database and a Database management system is because it is simply an open door to eventually understand data if Integrity is ever neglected. The definition of Integrity that I would like to focus on for this project is the idea that if someone were to gain access to the files that the data is stored on and were to have permissions to edit them, they could maliciously change a 1 or a 0 in the database without anyone knowing and therefore change what the data was. This quite easily would mean that an attacker could change a 10 to a 245 using simple bit flipping techniques such as the one below from a simple python repl example.



```
>>> 10 ^ 255                          >>> bin(10)
245                                   '0b1010'
>>> 245 ^ 255                         >>> bin(245)
10                                    '0b11110101'
```

Figure 1: Flipping Bytes Example

It would be very difficult to say that no one will have access to the binary files, or that the server is 100% secure. Therefore, the thing to do would be to prepare for any changes to the binary files or to the transferred data. This would be through first detecting that something has been changed in the originally encrypted data, and before using it as real data, understanding that the data is not to be trusted. This would obviously require a backup of some sort.

2 Related Work

This type of encryption that monitors and doesn't allow for changes in the encrypted data is currently available and used. I was able to see that one Python package, Fernet, uses the following standard cryptographic primitives.

- AES in CBC mode with a 128-bit key for encryption; using PKCS7 padding.
- HMAC using SHA256 for authentication.
- Initialization vectors are generated using os.urandom().

I was unable to make any sort of changes without the decryption breaking due to private key decryption errors. A very important limitation was stated, "Fernet is ideal for encrypting data that easily fits in memory. As a design feature it does not expose unauthenticated bytes. This means that the complete message contents must be available in memory, making Fernet generally unsuitable for very large files at this time."

Prior work that I would like to extend is a GitHub repo that uses AES-256 CBC mode encryption to combat bit flipping attacks that could be used on user roles such as "USERS" and "ADMIN" to maybe change the user's role to "ADMIN". I would like to extend this to a real DBMS. I think it would be interesting to see if user roles could be manipulated but I think that that goal will be to modify any data in the database. This was ultimately unable to be attempted.

The novelty or importance of this project is to first see if this will work and if there are any real vulnerabilities in these DBMS's encryptions. And second, if time permits, create a software that could be given some endpoints and file locations to test the integrity of any database or file system. I know that many popular databases such as DynamoDB and Firestore save their data on S3 buckets or GCP buckets and I think it would be interesting to see what can be found about their databases given Admin credentials to the database files.

3 Results

One problem or fault that I came to understand that was not caught in the Midterm Progress Report, was that I built my project to not just flip bits but to flip all bits within a byte. I think that I just did not have enough experience with this before I started my project to notice this important distinction in what a bytes array actually is. I think I understood the difference between bit and bytes, but when looping through bytes I expected it to be enough and didn't think that I should be flipping one bit at a time in each of those bytes. With this, I was unable to correct the error in time for this report but I think that project is still 100% successful and I learned and did a lot. The word bit(s) in all output therefore actually means byte(s).

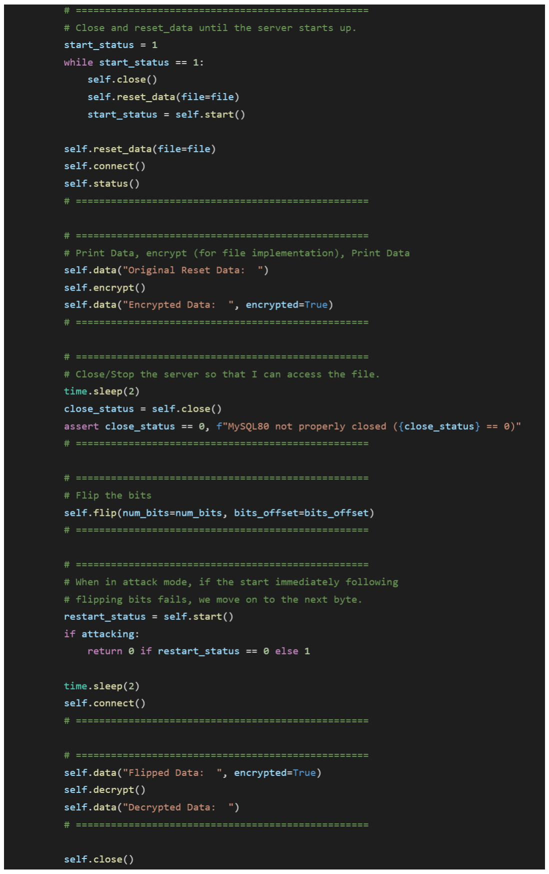
First I'd like to go into the results that I was able to achieve with MySQL. Then I'd like to go over an introduction to the software implementations that I was able to do for this project. Here we will see the things needed and problems I ran into when implementing the MySQL version of this project and also things about a simple File implementation. From there I'd like to go over things done for the basic file implementation and things I learned there. Lastly, we will go over some possible future work to go on top of this project and some solutions to preventing this type of attack.

MySQL is a great and famous DBMS, with MySQL I was able to achieve some sort of success with the project. There are two key achievements or outcomes of my work on MySQL:

- First, I was unable to modify any data in secret. Now I know that not modifying data wasn't my goal, but it does show that the MySQL server's encryption and integrity protection is working well. With that being said, due to compute time on my laptop I wasn't able to flip all bytes in the main.ibd file. This was because the file is 114688 bytes long and I needed to start and stop the server for each byte that was attacked. It would have taken a lot longer had it been 114688 x 8 bits and I flipped them one at a time.

- Second, A specific run failed when the server was unable to start back up after having flipped a byte. I was able to start the server after having flipped bytes, in three locations. These locations are byte indexes in the main.ibd, they are [35, 36, 37]. This is significant because I was unable to start the server back up after flipping bytes in any other location that I tried, which were as follows, [0-1981, 10000-15312, 50000-50200], out of the 114688 bytes in the file.

Here I would like to go into the code/software of my project. While I tried to focus on getting the software to work with MySQL, I also did a substantial amount of work on encrypting, flipping and decrypting basic files. The code that I have built could easily be modified to work on any database or file structure. I started with building an Abstract Database class that could do these tasks:

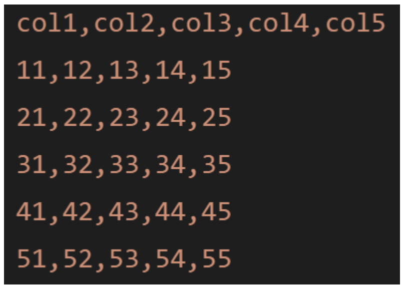


```python
# ==================================================
# Close and reset_data until the server starts up.
start_status = 1
while start_status == 1:
    self.close()
    self.reset_data(file=file)
    start_status = self.start()

self.reset_data(file=file)
self.connect()
self.status()
# ==================================================

# ==================================================
# Print Data, encrypt (for file implementation), Print Data
self.data("Original Reset Data:  ")
self.encrypt()
self.data("Encrypted Data:  ", encrypted=True)
# ==================================================

# ==================================================
# Close/Stop the server so that I can access the file.
time.sleep(2)
close_status = self.close()
assert close_status == 0, f"MySQL80 not properly closed ({close_status} == 0)"
# ==================================================

# ==================================================
# Flip the bits
self.flip(num_bits=num_bits, bits_offset=bits_offset)
# ==================================================

# ==================================================
# When in attack mode, if the start immediately following
# flipping bits fails, we move on to the next byte.
restart_status = self.start()
if attacking:
    return 0 if restart_status == 0 else 1

time.sleep(2)
self.connect()
# ==================================================

# ==================================================
self.data("Flipped Data:  ", encrypted=True)
self.decrypt()
self.data("Decrypted Data:  ")
# ==================================================

self.close()
```

Figure 2: Running Steps

- run, takes in the number of bytes to flip starting at a provided byte. This was the core method that would follow these instructions to make an attempt/attack against the server. As we can see in Figure 2

- data, prints out the data from the Database. If the database is off or it breaks when loading it displays the error message. If the Database is a simple file implementation this prints the plaintext of the file. This is what the rows and columns were in all databases. As seen in Figure 3

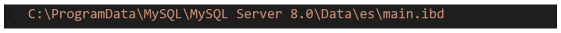


```
col1,col2,col3,col4,col5
11,12,13,14,15
21,22,23,24,25
31,32,33,34,35
41,42,43,44,45
51,52,53,54,55
```

Figure 3: Data

- reset, Resets the data back to original by deleting all old files and data.
- encrypt, Used primarily on the File implementation to encrypt the file with different types of encryption
- decrypt, Used to decrypt the File that was encrypted by the encrypt method
- flip, the core method to flip the bits of a given file at a given location and flips a given number of bits
- start, Starts the MySQL server and is not used for the File implementation
- stop/close, Stops the MySQL server and is not used for the File implementation
- connect, Attempts to connect to the Database and will display an error message is it fails
- attack, uses the run method to flip one byte at a time through all bytes in the file. This is the main method used to get results.

The MySQL implementation used the file located at:

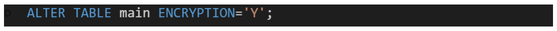


```
C:\ProgramData\MySQL\MySQL Server 8.0\Data\es\main.ibd
```

Figure 4: Data file

This is where all data for the database ‘es’ and the table ‘main’ was saved by the MySQL server. One key problem that I ran into was PermisionError. So, Even though I was pretending to be an Admin I still needed to turn off the MySQL server to get write privileges to the data files.

Adding encryption to this file was not difficult just by using:

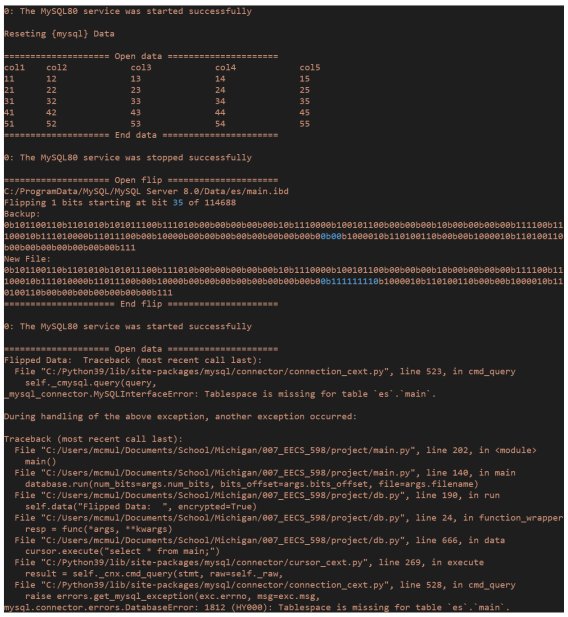


```
ALTER TABLE main ENCRYPTION='Y';
```

Figure 5: Alter Table

After encryption was added there was nothing different with the program/system because the MySQL DBMS managed the encryption and decryption completely. I only noticed that the bytes were different in the main.ibd file after ALTERing the database. The next image is some of the output of the software when attacking the MySQL server, this was at byte index 35 where the server was able to start but the Tablespace was corrupted/broken. You can see that when the software flips the 35 byte it changes the 1’s and 0’s, this is highlighted in blue. Here you can see that the server was able to turn back on after the data was modified, and at the bottom you can see that when the data was requested there was an error thrown that shows that the Tablespace for the table ‘main’ in the database ‘es’ was broken. As you can see Figure 6

The File Implementation was my favorite part of this project. I feel that I’ve learned alot about what encryption can provide and I feel that I may want to use it more in my own projects. I have built a fully functional business and website with devices but none of it is encrypted and all

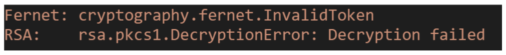


```
0: The MySQL80 service was started successfully

Reseting {mysql} Data

==================== Open data =====================
col1    col2            col3            col4            col5
11      12              13              14              15
21      22              23              24              25
31      32              33              34              35
41      42              43              44              45
51      52              53              54              55
==================== End data ======================

0: The MySQL80 service was stopped successfully

==================== Open flip =====================
C:/ProgramData/MySQL/MySQL Server 8.0/Data/es/main.ibd
Flipping 1 bits starting at bit 35 of 114688
Backup:
0b101100110b1101010b101011100b111010b00b00b00b00b00b10b1110000b100101100b00b00b00b10b00b00b00b00b111100b11
100010b111010000b11011100b00b10000b00b00b00b00b00b00b00b00b00b1000010b110100110b00b00b1000010b110100110
b00b00b00b00b00b00b00b111
New File:
0b101100110b1101010b101011100b111010b00b00b00b00b00b10b1110000b100101100b00b00b00b10b00b00b00b00b111100b11
100010b111010000b11011100b00b10000b00b00b00b00b00b00b00b00b0b111111110b1000010b110100110b00b00b1000010b11
0100110b00b00b00b00b00b00b00b111
===================== End flip =====================

0: The MySQL80 service was started successfully

==================== Open data =====================
Flipped Data:  Traceback (most recent call last):
  File "C:/Python39/lib/site-packages/mysql/connector/connection_cext.py", line 523, in cmd_query
    self._cmysql.query(query,
_mysql_connector.MySQLInterfaceError: Tablespace is missing for table `es`.`main`.

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:/Users/mcmul/Documents/School/Michigan/007_EECS_598/project/main.py", line 202, in <module>
    main()
  File "C:/Users/mcmul/Documents/School/Michigan/007_EECS_598/project/main.py", line 140, in main
    database.run(num_bits=args.num_bits, bits_offset=args.bits_offset, file=args.filename)
  File "C:/Users/mcmul/Documents/School/Michigan/007_EECS_598/project/db.py", line 190, in run
    self.data("Flipped Data:  ", encrypted=True)
  File "C:/Users/mcmul/Documents/School/Michigan/007_EECS_598/project/db.py", line 24, in function_wrapper
    resp = func(*args, **kwargs)
  File "C:/Users/mcmul/Documents/School/Michigan/007_EECS_598/project/db.py", line 666, in data
    cursor.execute("select * from main;")
  File "C:/Python39/lib/site-packages/mysql/connector/cursor_cext.py", line 269, in execute
    result = self._cnx.cmd_query(stmt, raw=self._raw,
  File "C:/Python39/lib/site-packages/mysql/connector/connection_cext.py", line 528, in cmd_query
    raise errors.get_mysql_exception(exc.errno, msg=exc.msg,
mysql.connector.errors.DatabaseError: 1812 (HY000): Tablespace is missing for table `es`.`main`.
```

Figure 6: MySQL Output

the data can be found by a few simple queries. I think that seeing the simplicity of how to encrypt using python I am motivated to use it more in my future software designs.

Some of the basic Out-of-the-Box encryption implementations in python are [Fernet, RSA, AES(MODE_CBC)]. I was able to implement these three encryptions and see if we can change any of the data when attacking each of them. Fernet and RSA worked as expected and threw errors whenever there were any changes to the encrypted file.



```
Fernet: cryptography.fernet.InvalidToken
RSA:    rsa.pkcs1.DecryptionError: Decryption failed
```

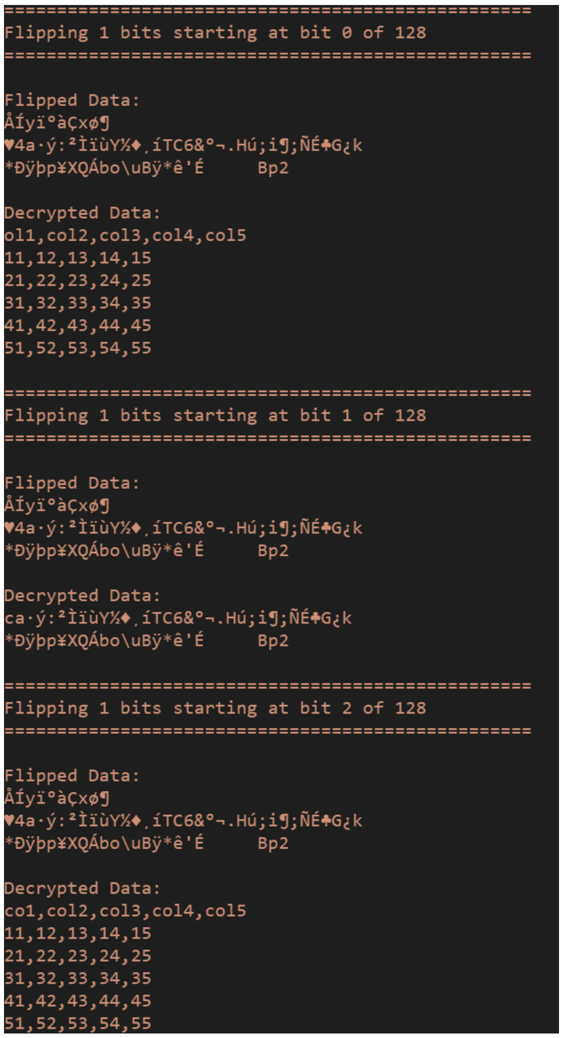
Figure 7: Fernet and RSA Errors

Fernet uses AES in CBC mode as mentioned above, with a few other modifications. This is important because I was able to implement my own AES encryption with CBC mode, an 'iv' and some padding. My implementation does not have PKCS7 padding, HMAC using SHA256 for authentication, or Initialization vectors are generated using os.urandom(). This proved to be a huge difference. As mentioned above I was unable to make any sort of change to the Fernet implementation and get away with it, but with my own AES with CBC mode implementation I was actually often able to get through the encryption without the encryption noticing. This makes me assume that the HMAC using SHA256 is what proves that nothing has been changed. It is interesting to see what can happen with my implementation. As you can see in the following output, I was able to actually drop bytes from the original data. As you can see, flipping the first encrypted byte gets rid of the 'c' in 'col1'. The second one actually broke the run and didn't allow it to be decrypted, but no errors were thrown, the plaintext was treated like it was the correct data. Flipping the third byte got rid of the 'l' in the column name 'co1'. As seen in Figure 8

4 Future Work

Ideas of Future Work to go on top of this project are:

- Finish running the software for all bytes in the main.ibd
- Change to flip bits instead of bytes
- Implement PostgreSQL, MongoDB, and maybe any other encryption technique that would like to be tested



```
==================================================
Flipping 1 bits starting at bit 0 of 128
==================================================

Flipped Data:
ÅÍyïºàÇxø¶
♥4a·ý:²ÌïùY½♦¸íTC6&°¬.Hú;i¶;ÑÉ♣G¿k
*Ðÿþp¥XQÁbo\uBÿ*ê'É      Bp2

Decrypted Data:
ol1,col2,col3,col4,col5
11,12,13,14,15
21,22,23,24,25
31,32,33,34,35
41,42,43,44,45
51,52,53,54,55

==================================================
Flipping 1 bits starting at bit 1 of 128
==================================================

Flipped Data:
ÅÍyïºàÇxø¶
♥4a·ý:²ÌïùY½♦¸íTC6&°¬.Hú;i¶;ÑÉ♣G¿k
*Ðÿþp¥XQÁbo\uBÿ*ê'É      Bp2

Decrypted Data:
ca·ý:²ÌïùY½♦¸íTC6&°¬.Hú;i¶;ÑÉ♣G¿k
*Ðÿþp¥XQÁbo\uBÿ*ê'É      Bp2

==================================================
Flipping 1 bits starting at bit 2 of 128
==================================================

Flipped Data:
ÅÍyïºàÇxø¶
♥4a·ý:²ÌïùY½♦¸íTC6&°¬.Hú;i¶;ÑÉ♣G¿k
*Ðÿþp¥XQÁbo\uBÿ*ê'É      Bp2

Decrypted Data:
co1,col2,col3,col4,col5
11,12,13,14,15
21,22,23,24,25
31,32,33,34,35
41,42,43,44,45
51,52,53,54,55
```

Figure 8: Fernet and RSA Errors

5 Conclusion

One solution that I have thought of for these possible attacks would be to maintain a hidden backup of some sort. The Fernet implementation throws an error when the decryption breaks. This could then trigger an immediate backup restore. The problem with this is if the attacker knows about the backup then they would attempt to flip bits in the backup. The reason that this is a good idea is because we have to assume that we don't know what could happen to the 1's and 0's, we have to pretend that anything can happen. In that case we have to be ready for anything.

In conclusion, I really enjoyed my time working on this project. I think that I was able to learn a lot throughout this project and I never thought I would be worrying about 1's and 0's but I loved learning about how data is handled and how software handles bytes and how bytes can be manipulated. MySQL does a great job at maintaining integrity protection and not allowing many changes to the data, if any were allowed then the tablespace was not loaded and threw an error. I think that my time spent understanding encryption schemes on regular files was so interesting and a great way to understand encryption and how I might use it in the future.

6 References

Fernet. (n.d.). Spec/spec.md at master · Fernet/Spec. GitHub. Retrieved April 24, 2022, from https://github.com/fernet/spec/blob/master/Spec.md

---

Fernet (symmetric encryption) - Cryptography 37.0.0.dev1 documentation. (n.d.). Retrieved April 24, 2022, from https://cryptography.io/en/latest/fernet/

Wikimedia Foundation. (2021, August 31). PKCS 8. Wikipedia. Retrieved April 24, 2022, from https://en.wikipedia.org/wiki/PKCS$_8$

RSA encryption implementation in python. Python Pool. (2021, May 31). Retrieved April 24, 2022, from https://www.pythonpool.com/rsa-encryption-python/

Journals, G. (2019, December 25). An analysis of flipped cryptographic block cipher mechanism for image. GRD Journals. Retrieved March 20, 2022, from https://www.academia.edu/41420042/An$_{A}nalysis_{o}f_{F}lipped_{C}ryptographic_{B}lock_{C}ipher_{M}echanism_{F}or_{I}mage$

Sat-based bit-flipping attack on logic encryptions. (n.d.). Retrieved March 20, 2022, from https://eprint.iacr.org/2017/1170.pdf

---

Bit-flipping attack. HandWiki. (n.d.). Retrieved March 20, 2022, from https://handwiki.org/wiki/Bit-flipping$_{a}ttack$

AES CBC bit flipping attack - youtube. (n.d.). Retrieved March 20, 2022, from https://www.youtube.com/watch?v=QG-z0r9afIs