1. What are some of the most egregious security issues with this file format?

The file format given in the specification is an attempt to store passwords in the form of key value pairs in an encrypted custom format database. There are few security issues with this database as follows:

- In order to read the database, a user input password needs to be provided. This password along with salt is hashed using MD5 and used a “Master Key”. This is a very bad design. Primarily for two reasons: (a) MD5 has proven instances of collisions i.e. cryptographically insecure and (b) But more importantly, MD5 is very fast. Which means an attacker can brute force the “Master Key” MD5 very quickly. The closest alternative to MD5 would be SHA1(deprecated) or SHA2 or even the new SHA3. In either cases, with enough computing resources these can be brute forced.
  The ideal solution in todays date would be using a Password Derivation Function like scrypt, bcrypt or PBKDF2.
- The database stores MD5 digest of decrypted unpadded value. Once again, as mentioned earlier this is an extremely bad idea. An adversary will never bother to brute force the key or password since the MD5 of decrypted unpadded value can be brute forced in a few minutes. (Entire english dictionary can be brute forced in few milli seconds for MD5 digest.)
  For this purpose, a better hashing algorithm is recommended. (e.g. SHA2)
- The “keys” in key value pairs are stored unencrypted. As the database is used for storing shared company passwords, the key will presumably contain usernames. And in case the database (though encrypted) is stolen, all the possible usernames (stored in “key”) will be leaked. And if those usernames contain dictionary passwords then it is only a matter of time before the passwords are guessed.

2. What are some of the most egregious security issues with this web application?

The web application provided uses flask micro framework and there are a few security issues such as:

- To login, the web application uses GET method instead of POST. When using POST parameters are sent in the HTTP body whereas, in GET these are sent in the URL. Hence, in the case of this web application the login is done using GET and the URL as seen by the user is http://localhost/show?password=test.

(Note: in either case of GET or POST, the password will be sent over the wire in plaintext. Hence best practice for login request is to use encryption over the wire i.e. TLS)

- The web application uses GET method to logout. http://localhost/logout. This can lead to a potential DoS for a targeted user. Consider a scenario where an user receives an image with tag <img src=“http://localhost/logout”></img>. This would logout the user off the site and will create a bad experience. Instead its advisable to use a DELETE method thereby ending the session.

- The web application asks for location on login page for analytics purpose and logs a lot data. This log can be misused and should be avoided.

3. What are some performance concerns that you have with this file format?

- Since, the file uses AES in CBC mode, encryption is not parallelizable. In other words, the entire database is encrypted together. So the IV used for quick password checking encrypts first 2 blocks and the next block is the value in key value pairs. Hence, in order to append a single new key value pair, entire database must be decrypted and encrypted again. This is ok for few key value pairs but as the number of entries increase, this will start costing a lot of time and resource. On the other hand, the same is not true for decryption as AES in CBC mode can be decrypted in parallel.