## Security Analysis

Enterprise database and information storage infrastructures, holding the crown jewels of an organisation, are subject to a wide range of abuses and attacks, particularly when left vulnerable by poor system design or configuration.

*We refer it to enhance our database application, we can do it like a questionnaire or checklist, if we consider the relative issue, mark it and get one score.*

**This checklist provide a preminary but efficient insights for users to access their own system.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| attack | suggestion | status | specific solutions | locations |
| Cloud Configuration Errors | 1. Knowing what data you hold (location, procedure,configuration) | √ | medical data | / |
|  | 2. Awaring that cloud services database are open to internet | √ | separate data server | Postgresql |
|  | 3. Strong authentication by default | √ | 2-factor | Java |
|  | 4. Implement monitor procedures on your cloud services | √ | log4j.jar | Java |
| SQL Injection(use supplied input without checking) | 1. Avoid the use of dynamic queries. Use prepared statements with parameterized queries. | √ | preparedStatement | Java |
|  | 2. Check users' parameters before using them. | √ | verify input data | Java + JS |
|  | \* bonus: monitor all dynamic queries in case someone bypass | √ | Log4j.jar + sql log | Java + Postgresql |
|  | 3. NoSQL databases should also be controled | X | no nosql db yet |  |
| Weak Authentication | 1. brute force control, blacklist&trail frequences. | √ | limit frequence | Postgresql |
|  | 2. Donnot encourage user to change password frequently. (but sometimes) | √ | record change time | Java |
|  | 3. Multi-factor authentication. | √ | 2-factor (password+ verifycode) | Java |
|  | 4. Donnot store password in plaintext. | √ | h(password, salt) | Java + Postgresql |
|  | 5. Strongly protect the application database credentials and make sure they are unguessable | √ | Store in safe enviroment and in ciphertext | OS |
| Privilege Abuse | 1. User access to data should be rate limited | √ | rate limit | Postgresql |
|  | 2. Not expose interfaces which allow arbitrary queries and bulk export of data. | √ | Only can access via stored procedure and function | Postgresql |
|  | 3. If must use bulk of data, log, audit, limited as few people as possible. | √ | only data owner can access their whole data | Java + Postgresql |
| Excessive Privilege | 1. Role based access controls | √ | Reader, writer, admin | Postgresql |
|  | 2. When staff change roles, their permissions are updated to reflect this | √ | if doctor are unauthorized by patient, he do not have read right anymore | Postgresql |
|  | 3. Regular, but not necessarily frequent, reviews the account status | X | No such concern yet |  |
| Inadequate logging and weak auditing | 1. What information you need to collect | √ | sql log and java log | Postgresql + Java |
| logging is the collection of data | 2. Consider how your logging data will be secured | √ | Protect logfile by OS | OS |
| auditing is someone actually looking at it | 3. Implement procedures for auditing the data collected so you know when something is amiss (displayed in a meaningful way) | √ | Simple IPS | Java + PG |
|  | 4. Implementing network-based audit appliances which monitor all database requests at a granular level and are independent of all users. | X |  |  |
| DoS | 1. Cloud based service protection | X | no cloud cluster yet |  |
|  | 2. Request rate limiting | √ |  |  |
| Exploiting unpatched services | 1. Maintain a complete and up to date inventory of the software components | √ | update server software (stable version) |  |
|  | 2. Establish a vulnerability management process (priority, assessment) | X |  |  |
| Insecure system architecture | 1. Boundary protection + depth defence | √ | Physical firewall + OS detect | OS |
|  | 2. Put internal data and external data into two parts | √ | Different data into different db, such as role and records | Postgresql |
|  | 3. Review the security of the management interfaces | √ | management interface are not exposed to internet in the same way like login page | Java + JS |
| Inadequate Backup | 1. Backups should be encrypted to protect confidentiality and integrity of the data | √ | pg\_probackup | Postgresql |
|  | 2. Online and offline backup (ransomware) | √ | RAID 5 + Tape | Postgresql + OS |
|  | 3. Resilience within cloud services | X |  |  |
|  | ( backups are not subject to the same threats as the live data and that full compromise of the live data environment cannot also compromise your backups. And do test your restore procedures, regularly.) |  |  |  |
| Other Good Practice | 1. Protect the RESTful api from random access via Backend authentication mechanism | √ | Springboot Security | Java |
|  | 2. Transparent encryption to protect data physically | √ | Pgcrypto | Postgresql |
|  | 3. Data transformed via SSL | √ | Url use SSL | Java |
|  | 4. Expose less port | √ | OS disables the unnecessary ports | OS |

We will find that we get a score : 31 / 37 and we can draw conclusions that,

1. If all of the above threats are considered and the appropriate recommended defensive measures are taken, your system will be quite safe when data are *in transit* and *at rest* process
2. Traditional security is not guaranteed to protect those data *in use*

## 