Computing Arbitrary Functions of Encrypted Data

Klev

September 15, 2015

1 Abstract

Two problems: Curious database managers that peaks into the information stored in the system, and malicious adversaries that obtains access to the system through bugs and security flaws. With physical access to the database, an attacker would be able to access all data, both in memory and what is stored on disk.

CryptDB: Provides practical and provable confidentiality when exploited to these types of attacks on applications containing an SQL database.

Does so by using a collection of efficient SQL-aware encryption schemes. Can bootstrap encryption keys to user passwords. Can only decrypt using the password of some of the users with access to the data.

Low overhead, reducing throughput by 14,5%. Which is modest.

2 Introduction

Approaches: Encrypt all data in the database/on the server. Let the client process the data. Kinda stupid. Puts a lot of competition on the user, and is useless for application websites/systems that process a lot of data. Difficult to convert to this kind of solution.

CryptDB: Intermediate design for applications using database management systems. Leverages the application server - database server model. Practical because of SQL uses a well-defined set of operators which can be supported efficiently.

Problem 2: Cannot guarantee security of users that are logged-in during an attack.

Two challenges:

1) Limit the amount of confidential information that is exposed to the application/system, and at the same time support a large amount of different types of queries to be performed.
If encrypting the data with a strong cryptosystem, say AES, you would defeat much of the purpose as you would have to give the system your decryption key for it to be able to locate and decrypt the information you need.

2) Minimize the information that is leaked if the system gets compromised. Ensure that the system can only obtain a limited amount of decrypted data. Giving each user a database encryption key does not work for shared data.

Three key ideas:

1) Execute SQL queries over encrypted data. Leverages that SQL is made of well-defined set of primitive operators. Encrypts the data in a way that makes it possible to perform these operations on the transformed data. Efficient because it uses symmetric key encryption, and avoids fully homomorphic encryption.
2) Adjustable query-based encryption. Uses an "onion" of encryptions to avoid leaking information. Adjusts the encryption layer in realtime based on the queries. Compactly store multiple ciphertexts within eachother.
3) Chain encryption keys to user passwords. Can only decrypt data using a chain of keys consisting of a user with access' password and an encryption key.

3 Security Overview

Intercepts all SQL queries using a Database Proxy server. the function of this proxy is to rewrite the query to be able to operate on the encrypted data. Encrypts/decrypts data.

The database never receives decryption keys, so it is not capable of decrypting any of the information stored.

Developers must annotate the SQL schema to define different principals. Their keys will be able to decrypt the different parts of the database.

Out of the scope: Deleting the database after getting access, SQL injection and XSS.

Goal of CryptDB: Confidentiality, not integrity or availability.

Attacker assumed to be passive: Does not change queries, results or the data in the database system.

Database uses a secret key to encrypt all data inserted in the DB + queries.

SQL-aware encryption that dynamically adapts to the queries asked from the user. Examples: Group by, Sort, Max and Min

Reveals the overlaying table structure of the database, but not column names or types, number of rows and size.