Inference Firewall System (IFS)

ISO/IEC/IEEE 42010:2011

Systems and software engineering - Architecture Description

Supervisor:
Jason Jaskolka

Team Members:
Aleksandar Savic - 100999110
Hasan Issa - 100921446
Ryan Zheng - 100996797
Calvin Soong - 100999332
Tashfiq Akhand - 101004428

Table of Contents

1 Introduction

IFS is a web application available for any provided database and the respective users that perform queries on that database. Its purpose is to block possible inference attacks on the database through the use of preset administrator defined database policies. The IFS application interacts with a DBMS through the use of existing APIs in the backend and shall only be running if the database and web server are active.

The following sections of the document give a high level design ~~overview~~ of the IFS system. It will Identify and explain the overall architecture of the system, and provide the reasoning and justification of the design architectures chosen. This document will also Identify the main subsystems/components of the system, and provide a brief description of each subsystem.

1.1 Scope

This software system to be produced is the Inference Firewall System (IFS). IFS aims to provide database users with more advanced security features to prevent any sensitive data from being exposed to unauthorized personnel or intruders, but also from authorized users. The proposed system is a web interface used to provide access control to the database at hand and manage the level of exposure to sensitive data depending on the set security clearance role of an authorized user. The system itself does not insert or modify any data in the database itself, it is merely an interface for users to query data, and provide inference free results. The main objective of this system is to prevent inference attacks on the targeted database through the use of preset database policies and stored query logs.

1.2 Definitions, Acronyms, and Abbreviations

IFS	Inference Firewall System, which is this application name.
API	“Application Programming Interface”. A set of functions and procedures allowing access to the features or data of an operating system, application, or other service.

2 Architecture Descriptions

2.1 Introduction

The architecture for the entire system, the IFS, would best be a pattern that allows communication between the frontend, backend, and the middleware, known as the Inference Detection Subsystem. The Broker architecture pattern is what will be used to test the system including the frontend and backend with the main subsystem acting as a broker in this situation. The broker pattern would be most suitable since the broker, which is the middleware, would be the mediator between the client/user and the database subsystem. This allows efficient communication throughout the entire system since it will run as a client-server system that optimizes communication, and communication is important since the middleware needs the admin information of the client to check it, see if it adheres to the database policies, make a decision in terms of the next action to take, and then grant the client permission to access the targeted database. This operation requires the communication between three subsystems and the broker pattern would be most suitable since it receives messages and sends responses, as well as bringing advantages such as performance, scalability, and reliability.

The subsystem that we are most interested in is divided into many key layers where the layer pattern will be used for maximized efficiency and organization. Layered architecture allows you to distinguish and distribute the responsibilities that the application has to deliver to the end user. The Inference Detection Subsystem has the layers organized into three layers. First the Interface layer, then the Logic layer, and then the Data layer. The reason for this order is that the Interface layer would be the mediator for the whole system receiving client user info and then passing it down to the Logic layer. The logic layer performs a number of operations on the info, and then sends it to the Data layer. The data layer checks the policies/access rules and prepares a response for the logic layer that will either deny or authorize access to the database. The logic layer will then forward the response to the interface layer, which is SQL Wrapped for Database use.

The Subsystems/Components responsibilities for the IFS would be:

<u>The User Demo Interface component</u>

- A demo system used by users query data from the database
- Allows administrators to add, delete, and modify Database Policy Validation rules
- Users can login to demonstrate the algorithm's ability to differentiate the query results based on the user's security level
- HTML/CSS/JavaScript pages served by Node web server with Express middleware
- Connects to the Inference Detection Subsystem via Web API

Inference Detection Subsystem

Interface

- Web API
- Receive and Send Data
- SQL Wrapper API to talk to the Database Interface Subsystem

Logic

- The Controller, decides what action the system should take next based on the current state of the system
- Decision Engine (Database Policy Validation) decides whether the query violates a policy based on the current log of queries, and what action to take in regards to the results

Data

- Query Log Component, contains all the queries made
- Configuration File, contains the security levels, user names, passwords, and roles of all the users. Made by the system administrator
- Set of Policy Rules, contains the database policies to prevent inference attacks. Made by the system administrator

DBMS

- Stores the sample database columns and data

2.2 Architecture description identification and overview

The overall architecture of the system to be tested will be a broker pattern where the IFS will act as a broker interface between the web interface for the user and the database to be tested. The IFS will receive data from the user, query the database, manipulate the data and send it back to the user. The broker pattern is the most optimal architecture for this system due to the heavy amount of client requests and the need for an extra layer to be placed on top of the DBMS to modify data accordingly. A diagram of the architecture can be seen below.

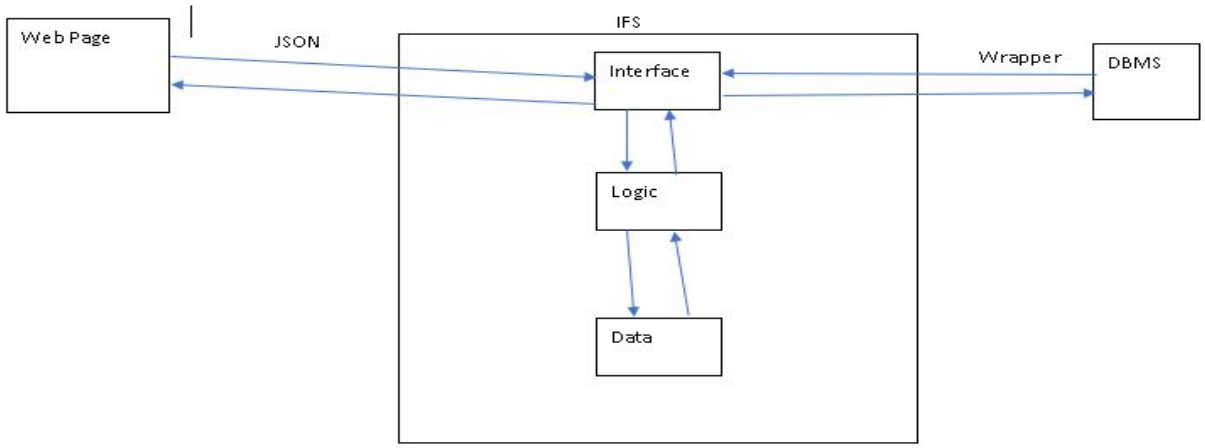

Figure 1: Broken pattern for system to be tested.

The architecture to be used for the system of interest (IFS) will be the layer pattern consisting of the interface layer, logic layer and data layer with its respective components as shown below. The data will be received through the interface layer and manipulated in the logic layer according to the components that are present in the data layer. The layer pattern is the optimal architecture for this system because it creates an organized structure for the system to handle data received through the external environment before it is sent back. Please refer below for the layer diagram of the IFS system.

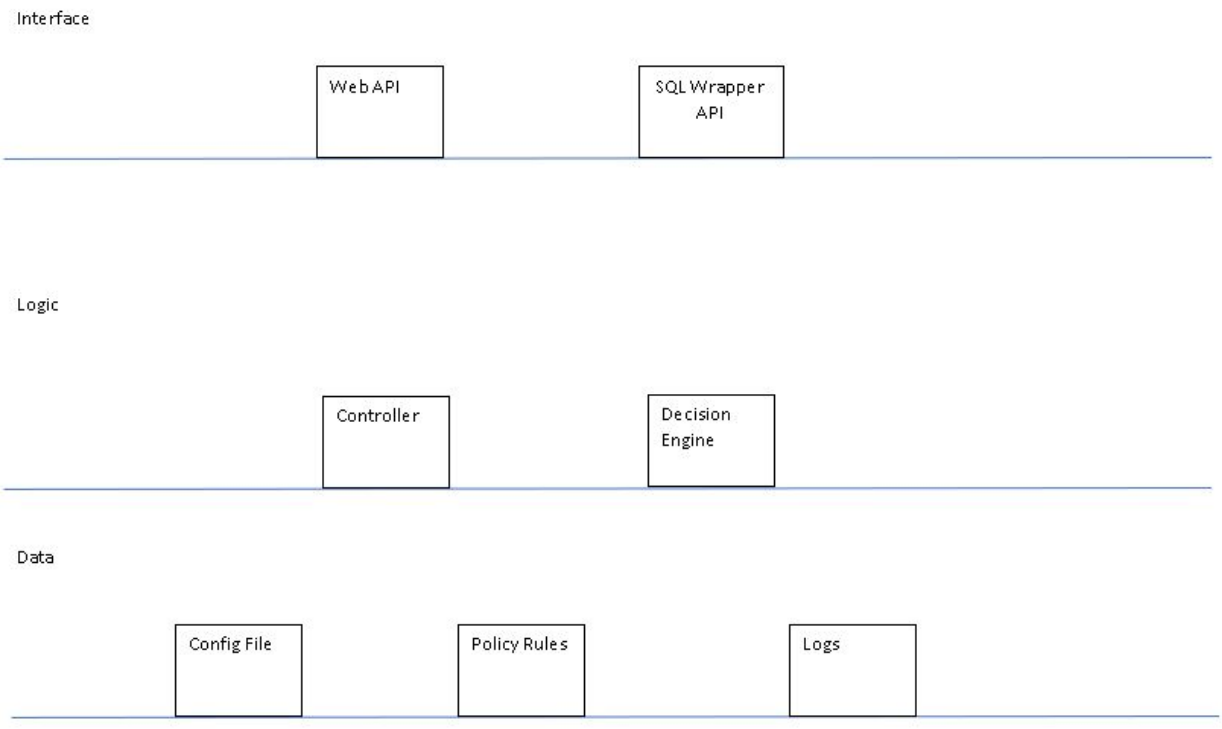

Figure 2: Layer pattern of the IFS system.

2.3 Identification of stakeholders and concerns

The stakeholders for the system includes the company (database owner), system administrator, and the authorized users that query the database.

The company is the acquirer of the system. The company is concerned about the difficulty to incorporate the system between their existing database and their user interfaces that are used to query the existing data. The company will be concerned about the potential performance impacts that the system will introduce to the database in regards to querying time, as well as potential reliability issues that may arise due to the addition of the IFS system.

The system administrator is the administrator of the database. The administrator is concerned about the maintainability of the system and the ease of configuring database policy rules, and setting up the configuration file.

The authorized users are users with authorized accounts created by the administrator who can query data from the database. These users will have roles and security levels assigned to their accounts by the administrator. The users are concerned about the usability of the database after the system is introduced. As the users may accidentally trigger an inference attack, the system should not give the user an error message and prevent them from using it. The users are also concerned about the performance impact that the system will introduce to the database querying time.

2.4 Architecture model

The entire system uses the Broker Pattern as the abstraction of the overall system. Broker Pattern is often used in a distributed system with decoupled components. The broker architecture opens the coordinating communication between client and server, hence the requests and transmitted results can be filtered in the broker. The broker ensures the server and client retain their own functionality but add a layer of data security.

The broker component itself (IFS) uses the layered architecture pattern which is used for maximum efficiency, flexibility, scalability, and organization. Layered architecture allows you to distinguish and distribute the responsibilities that the application has to deliver to the end user.

The User Demo Interface is a series of web pages created using HTML, CSS, and JavaScript. Currently, 88% of websites use HTML and 79.9% use HTML5 [1]. This ensures that the demo interface will be similar to real user interfaces used by modern companies.

The User Demo Interface will be hosted using a local Node.Js Server. The Node server makes hosting the web pages much easier as it is written using JavaScript, and the express middleware is used to handle requests from the client.

The IFS will be written using Java due to Java's ability to work with databases and socket programming support. Java provides an API for accessing and processing data from SQL commands and the sockets can be used to communicate with the Node.Js server.

2.5 Architecture relations

The inference detection system will communicate with the other systems and its surrounding environment through the inference layer. Requests from users on the web page, including the admin, will be sent and received as JSON objects using the Web API. To interact with the DBMS, the system will use SQL wrappers to execute and retrieve the queries made by the user.

The data layer will consist of the configuration file, policy rules and the query logs. The configuration file will store data such as usernames, passwords, the database to be queried and the user roles/security level. The query logs made by the user will be stored in the data layer to keep a history of the user's queries and will also be used by the decision engine to ensure no sensitive data is vulnerable by inference attacks from the previous queries. The policy rules set by the administrator will also be stored in the data layer and be used by the decision engine to ensure no queries are in violation of the confidentiality policy.

The logic layer will consist of the decision engine and the controller. The controller decides what action the system should take next based on the current state of the system and will handle requests from the interface layer. The decision engine will be the mediator between the user's requests and the data layer. JSON objects will be parsed through the decision engine and important data including the logs, configuration features and policy rules will be stored in the data layer. The decision engine will also grab the queries made by the user and communicate with the policy rules to ensure the queries adhere to the confidentiality policy set by the administrator and respond accordingly using the java algorithm based on polyinstantiation and cell suppression.

2.6 Architecture rationale

The layer architecture pattern is the best choice for IFS due to flexibility and reusability. The interface layer in IFS allows multiple applications to use the Web API and multiple databases can be supported simply by using database wrappers. The logic layer and the data layer will not need modifications when the interface layer is changed.

When including the user interface and the database, the IFS is a broker architecture. This allows the system to be easily integrated for existing databases and user interfaces. By using the Web API and the SQL wrapper provided in IFS, IFS can be installed by pointing the user interface to the Web API and the database to the SQL wrapper.

Other architectural patterns were not used for the sole purpose of effective functionality. The layer pattern would build effective organization within the IFS and the broker pattern would be the most practical system for the whole IFS, including the user interface and the database, in order to strive for optimal performance, scalability, and reliability. Other architecture patterns that were not selected were Model-View-Controller (MVC) pattern, Pipe and Filter pattern, Peer-to-Peer pattern and Service-Oriented pattern.

The Model-View-Controller wouldn't be sufficient due to problems that MVC arises which are complexity and high coupling since all components are dependant on each other for functionality. The Pipe and Filter pattern is not a good choice since it is not great for interactive systems which the IFS is, and many filters aren't needed which results in high computation overhead. The Peer-to-Peer pattern would not be good due to managing security, data consistency, data/service availability, backup, and recovery are more complex, and the IFS requires those factors to be fully functional and effective. Lastly, the Service-Oriented pattern was not a great candidate due to the issues that it has on having performance overhead associated with the middleware which is the most important component.

Therefore, the layer pattern is best suited for the IFS and the broker pattern is great if the User Demo Interface component and the DBMS is involved with testing in the overall system.

References

[1] W3techs. *Usage Statistics and Market Share of HTML5 for Websites, October 2019*. [online] https://w3techs.com/technologies/details/ml-html5/all/all

Contributions

Calvin

- 2.3, 2.4, 2.6

Ryan

- 2.4 and front-end set up

Aleksandar

- 2.2 and 2.5

Hasan

- 1 and 2

Tashfiq

- 2.1 and 2.6

Calvin Soong 

Ryan Zheng 

Aleksandar Savic 

Hasan Issa 

Tashfiq Akhand