**4.1 Introduction**

In this chapter, we present our detection model using the BERT language model to prevent from the SQL injection attacks. We begin by laying down BERT's architecture and the reasons behind choosing it for this task. We then move on to explaining the preprocessing that must be done in order to make our dataset compatible with BERT's input requirements, and then move on to explaining the fine-tuning process employed in order to train the model for binary classification.

**4.2 General Conception**

The deep learning-based model introduced in this work will detect SQL injection attacks. The architecture of the model is built upon BERT (Bidirectional Encoder Representations from Transformers), and has been fine-tuned for binary classification using labeled SQL injection and benign SQL queries.

**4.2 Why BERT for SQL Injection Detection**

We selected BERT as our SQL injection detection model due to its high capacity to comprehend both meaning and context of textual information, particularly in structured input such as SQL queries. What makes BERT distinctively effective is the fact that it processes the whole input bidirectionally which helps it pick up on subtle patterns that might indicate an attack. Unlike older models that generally rely on basic keyword matching or predefined rules, BERT learns from the actual composition and intent of the query. After fine-tuning on a database consisting of normal and malicious SQL queries, the model was able to accurately identify suspicious inputs that could potentially represent injection attempts. Thus, BERT proves to be a reliable and effective tool for web application security improvement.

**4.3 BERT Architecture Overview**

The BERT model architecture is a multilayer bidirectional Transformer encoder, based on the encoder component of the original Transformer model. While the original Transformer use an encoder-decoder structure where the encoder processes input sequences using self-attention mechanisms and the decoder combines self-attention with encoder-decoder attention, BERT focuses exclusively on the encoder to learn deep contextual representations.

The BERT architecture is available in two main configurations : BERTBASE and BERTLARGE. BERTBASE consists of 12 encoder layers, 768 hidden units per layer, and 12 attentions heads, resulting in approximately 110 million parameters. In contrast, BERTLARGE extends the architecture to 24 encoder layers, 1024 hidden units, and 16 attentions heads, totaling around 340 million parameters.

As a result, BERT models are capable of learning more complex contextual representations from input sequences.

(hedi tbenli ma ndirouhech ) {The input to the model begins with a classification token, [CLS], followed by a sequence of words. The [CLS] token is exclusively utilized for classification. The input is then fed into the sequence of encoder layers, wherein each is executing self-attention mechanisms followed by feedforward neural networks, and then to the subsequent encoder layer.

The output of the model is a 768-dimensional hidden vector for BERTBASE. In classification, the [CLS] token's output is typically used as the representation of the input sequence and passed through a classifier layer. helps generate more detailed language representations which specifically benefit tasks dealing with syntax-sensitive input such as SQL queries.}

We decided to use the BERTBASE model for our implementation mainly because our dataset contains 30,614 labeled queries. Since BERTBASE is less complex than BERTLARGE, it trains faster and uses fewer resources, while still performing well for classification tasks like detecting SQL injection attacks.

{ To adapt BERT for the task of detecting SQL injection attacks, we performed fine-tuning using a dataset labeled with both benign and malicious SQL queries. In this stage, the model was trained to classify each input as either safe or potentially harmful. Rather than retraining the entire network from scratch, we kept the core BERT layers intact and focused on updating the final classification layer. This approach allowed us to take advantage of BERT’s deep language understanding while tailoring the model specifically to the patterns found in SQL injection attempts.}

**4.4 Development Environment Overview**

**4.4.1 Programming language**

**Python**

Python is widely used in machine learning and deep learning, and for good reason. Its clean and simple syntax makes it easy to learn and work with, which is especially helpful when building and testing complex models.  A key strong point of Python is its extensive array of libraries, which include TensorFlow, PyTorch, Scikit-learn, and Keras. These libraries offer pre-built components that save time and simplify the development process, making the development process easier. Python also has a huge community of developers and researchers, which means there’s plenty of documentation, tutorials, and support available online. Python offers simplicity, flexibility, and strong tools, all of which render it an optimal language for machine learning and deep learning projects.

**4.4.2 Libraries**