# Anomaly Detection in Database Operating System

by

Brian Xia

B.S. Computer Science and Engineering
Massachusetts Institute of Technology, 2021

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022



Author .....
Department of Electrical Engineering and Computer Science
May 6, 2022

Certified by .....
Michael Stonebraker
Adjunct Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

# Anomaly Detection in Database Operating System

by

Brian Xia

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

Database Operating System (DBOS) is a new operating system (OS) framework that replaces the traditional file-based system with a high-performance database management system (DBMS). This design choice addresses the needs of a rapidly evolving software and hardware landscape that cannot be met by a traditional, mainstream OS. However, DBOS is a relatively new project under active development, with some missing secondary capabilities. In particular, the provenance capture system has not been fully explored with respect to real-time anomaly detection. To that end, Nectar Network (NN) was developed on top of DBOS as a public web application to generate real-world traffic and provenance data. In this thesis, I present a machine learning (ML) model to label anomalous provenance data captured by the NN, in the form of HTTP logs, in real-time. The model consists of two components: tokenization and classification. In the tokenization step, Byte-level Byte Pair Encoding (BBPE) breaks down the input bytes into token bytes that hold semantic meaning. In the classification step, a Convolutional Neural Network (CNN) takes the token bytes as input and outputs the predicted probability of anomaly. The model achieved strong performance, with a F1 score of 0.99951. Importantly, this work serves as a proof-of-concept for future endeavors to develop real-time security analysis features on top of DBOS systems.

Thesis Supervisor: Michael Stonebraker
Title: Adjunct Professor

# Acknowledgments

First and foremost, I would like to thank my thesis supervisor, Michael Stonebraker, for providing me the unique opportunity to work on this research project. His insights and feedback have been instrumental in the overall vision and direction of this work. I would also like to thank Çağatay Demiralp and Deeptaanshu Kumar for our weekly meetings and their technical support. Finally, I would like to thank Qian Li and Peter Kraft for their continuous support and guidance throughout the year.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The introduction begins with a brief discussion of the shortcomings faced by many mainstream OS. I then introduce DBOS, a new OS framework that utilizes a DBMS to address these shortcomings. Given that DBOS is a fairly new framework, its security features are underdeveloped, despite a robust data provenance capture system. The following section discusses NN, a web application designed on top of DBOS to generate real-world provenance data. The final section highlights the main contributions of this project, primarily the deployment of a machine learning model to actively and accurately tag NN provenance data as anomalous or benign.

## 1.1 Current operating systems

Hardware platforms have changed drastically throughout the years, with innovations made in both the design and integration of various components [2]. Noticeably, many mainstream operating systems have remained relatively untouched since their inception. At the time, these OS were designed to handle hardware platforms that consisted of a single processor and limited main memory, with a small set of executable functions [3, 4]. Hardware platforms today may now consist of hundreds of thousands of processors and multi-level memory, with a large and diverse set of services being requested by a multitude of users [5, 6, 7]. This significant increase in resources that need to be managed and scheduled poses several concerns for current, mainstream

OS architectures:

1. **Heterogeneous hardware**. Achieving optimal performance on a wide range of applications is often not feasible with only a standard CPU. It usually becomes necessary to bring in specialized hardware (GPUs, TPUs, SSDs, FPGAs, etc.) with diverse processing capabilities to handle specific tasks [6, 8, 9].

2. **Novel applications**. Many of the specialized hardware mentioned above are driven by the needs of novel applications. In particular, data-centric workloads, such as machine learning [9] and "big data" applications [10], are quickly becoming a cornerstone of ongoing research and innovation. Concretely, the tensor processing unit (TPU) was developed by Google with the specific aim to optimize neural network machine learning on TensorFlow software [11].

3. **Data privacy**. The rise of digitization and the internet has led to a massive influx of personally identifiable information (PII). There has been an ongoing shift from written to digital records in many settings, including clinical trials, hospital records, educational reports, etc. [12]. Social media has also been a major source of PII that is particularly sensitive given that many users are underage. Perhaps most famously, General Data Protection Regulation (GDPR), an EU law on data protection and privacy, established guidelines on PII processing [12, 13]. One important clause specified in GDPR is the "right to be forgotten", meaning that a user can request for all of their data to be completely removed from a given database. This is directly addressed by data provenance, which tracks the inputs, entities, systems, and processes that contributed to the data of interest. However, many OS lack integrated support for data provenance given that PII was not a primary concern or consideration when they were first being developed.

## 1.2 Database Operating System

DBOS is a joint effort by MIT, Stanford, CMU, Google, and VMware to leverage a distributed, transactional database management system as the foundation of a novel OS stack [1]. The underlying distributed, transactional DBMS helps to combat issues that result from today's massive scale systems. The DBOS environment stores the OS and application states within the structured DBMS, thereby allowing for fast updates and searches through simple SQL queries. Support for heterogeneous hardware is continually being added directly into DBOS through processor-specific stored procedures. Several data-centric workloads have been experimentally shown to exhibit higher performance running on DBOS than on comparable serverless environments such as Openwhisk [14] and Boki [15]. Data privacy has not been directly implemented within DBOS, but integrated functionality for data provenance capture should facilitate future endeavors to do so [16].

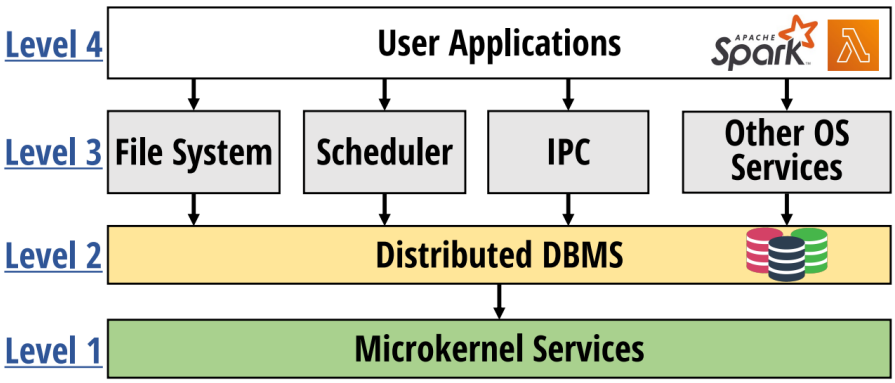



Figure 1-1: DBOS stack. Taken from [1].

The DBOS stack consists of four layers as shown in Figure 1-1. The top level consists of user applications that run protected from each other and any other levels. Notably, the ease and transparency with which users can query for process IDs, memory usage, or other related metrics greatly enhances the development of distributed applications. The next level consists of OS services such as distributed filesystems, task schedulers, interprocess communication and others. These services benefit greatly from the lower level distributed DBMS, which provides high availability, transaction

support, security, and dynamic reconfiguration. The following level consists of the distributed DBMS. DBOS employs VoltDB [17], which is a commercially available SQL database that offers low latency distributed transactions (concurrency control), synchronous replication (high availability), and ACID compliance. The bottom level consists of microkernel services such as raw device handlers, interrupt handlers, and basic inter-node communication.

### 1.2.1 Provenance capture

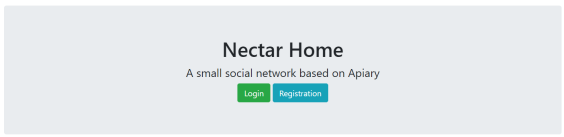
The DBOS stack is well designed for supporting a robust data provenance system. A high-performance DBMS can store structured, provenance information that is easily accessible through SQL queries. In practice, VoltDB [17] serves as the main memory DBMS that stores OS state. However, VoltDB is unsuited for handling large amounts of provenance data, requiring data to be spooled to Vertica [18] for long-term storage and downstream analysis. Kumar et al. have demonstrated that Vertica outperforms VoltDB dramatically with regard to provenance queries on tables with greater than $10^5$ rows [16]. This clearly highlights the importance of a dedicated OLAP system like Vertica to serve as the storehouse for provenance data.

One important point of discussion is what can be answered through such a data provenance system. The following list documents some potential queries of interest.

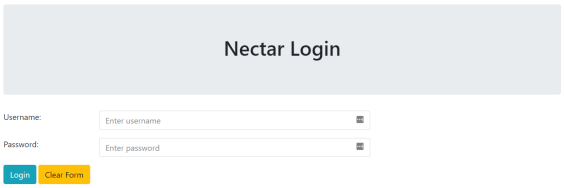
- **Table history**. Who was the last person to write to a particular table? Which table had the most updates over an arbitrary time frame?
- **Compromised users**. What are all of the blocks that were read or written by a compromised user over an arbitrary time frame? Who are all of the users that read a compromised block over an arbitrary time frame?
- **Chain of provenance**. What are all of the blocks that may have resulted from reading a particular block (downstream)? What are all of the blocks that may have influenced a particular block (upstream)?
- **Debugging**. What is the exact state of a table at a particular point in time?

- **PII**. Can you determine whether a particular block is legal based on the legality of the blocks used to generate it?

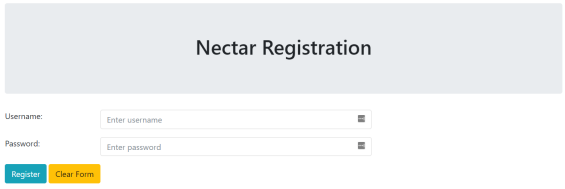
### 1.2.2 Nectar Network



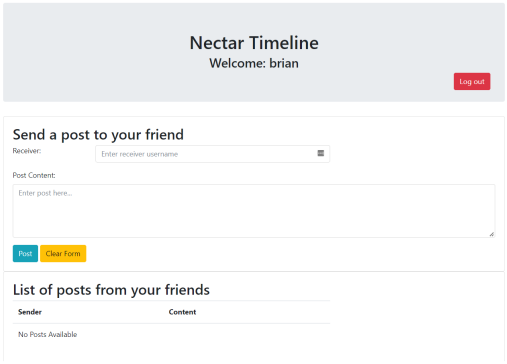

(a) Home page.




(b) Login page




(c) Registration page




(d) Timeline page

Figure 1-2: Nectar Network pages.

Nectar Network is a simple web application developed on top of DBOS. It serves as a rudimentary social networking site and is publicly accessible at nectarnetwork.org. All of its available pages are shown in Figure 1-2. The home page consists of two labeled buttons that redirect to the login and registration pages. The login page is a simple form that prompts the user for a username and password in the text boxes. There are also buttons that allow the user to submit a login request or clear the text boxes. The registration request contains the same layout as the login page except that the login button is replaced by a register button. When a user has successfully

logged in, they are brought to the timeline page which contains two parts. The top part allows a user to send a post to another user, and the bottom part displays all posts received by the user in a table format in which the first column is the sender username and the second column is the post content.


```
{
  "getAuthType" : null,
  "getPathInfo" : null,
  "cookie: JSESSIONID" : "D8B035F1CD70BC14F7E1804C54911F2B",
  "getRemoteAddr" : "171.66.11.171",
  "getServletPath" : "/home",
  "getMethod" : "GET",
  "getContextPath" : "",
  "getServerName" : "nectarnetwork.org",
  "getPathTranslated" : null,
  "getRequestedSessionId" : "D8B035F1CD70BC14F7E1804C54911F2B",
  "header: user-agent" : "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36",
  "getRequestURL" : "http://nectarnetwork.org/home",
  "header: upgrade-insecure-requests" : "1",
  "header: host" : "nectarnetwork.org",
  "getRequestURI" : "/home",
  "header: accept-encoding" : "gzip, deflate",
  "header: connection" : "keep-alive",
  "getQueryString" : "test=a",
  "getProtocol" : "HTTP/1.1",
  "header: accept" : "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
  "header: dnt" : "1",
  "getLocalName" : "dbos-qian-web-voltdb-main-0.c.dbos-2b63.internal",
  "getRemoteUser" : null,
  "header: cookie" : "JSESSIONID=D8B035F1CD70BC14F7E1804C54911F2B",
  "getContentType" : null,
  "parameter: test" : "a",
  "header: accept-language" : "en,zh-CN;q=0.9,zh;q=0.8"
}
```


Figure 1-3: Example HTTP log request from NN.

NN was made publicly available in order to capture real-world internet traffic, thereby allowing us to test DBOS provenance capture and develop real-time anomaly detection using a realistic web application deployment. All HTTP requests are logged and captured in Vertica [18] using the fields shown in Figure 1-3. This schema loosely follows the W3C extended logging format as described by Microsoft [19]. This format contains enough information to form a complete history of an HTTP request.

## 1.3 Web application attacks

Web applications have quickly become one of the most popular platforms for information and service delivery [20, 21]. It boasts several features that have led to its success, namely remote accessibility, cross-platform compatibility, and fast development. Importantly, web applications have become increasingly used for sensitive information related to health or financial institutions. An investigative report by Verizon in 2010 discovered that web applications are the most common attack vector (means by which an attacker can gain access to a network server) used for unautho-

rized intrusion, resulting in the most breaches and compromised data [22]. In the following sections, I will detail the most common types of web application attacks, including the methodology and end goal.

### 1.3.1 SQL injection

A SQL injection attack occurs when a malicious user tampers with the SQL queries sent by the web application to its corresponding database [21, 23]. This occurs when SQL keywords or operators are inserted into queries without input sanitization to explicitly remove or filter them out. This can be done through malevolent insertions into user inputs (e.g. fillable fields), cookies, or HTTP headers. Importantly, the contents of the insertion dictates whether the attack is of the first- or second-order. First-order attacks are executed immediately with the intent to return results immediately. In other words, the entire attack is localized within the insertion. A concrete example is using the union keyword to attach malicious SQL queries to the end of standard SQL queries. Second-order attacks rely on an initial insertion that lies dormant for some period of time, usually until a follow-up insertion prompts the execution of the first insertion. A concrete example is inserting an initial malicious query that can be prompted at a later date. The follow up query would return metadata on users who have accessed the web application since the initial insertion. The overall purpose of these attacks may be to steal credentials, alter data, delete data, and/or access connected resources.

### 1.3.2 Cross-site scripting

Cross-site scripting (XSS) occurs when a malicious user is able to execute custom scripts in a victim's browser [21]. This typically occurs when web responses are unsanitized, meaning they are unchecked for special characters/keywords that may lead to unexpected or malicious behaviors. This becomes problematic when web applications utilize the same-origin policy, which allows scripts in one webpage to access the data in another webpage if they both come from the same origin (combination of

URI scheme, host name, and port number). For instance, an attacker could insert a malicious script into a less secure webpage in order to access confidential information from a more secure webpage. Similar to SQL injection, there are first- and second-order attacks that dictate the timing when the attack occurs. A first-order attack, such as reflected XSS, prompts the user to click on a custom link which delivers an XSS payload to the web application. This payload allows the attacker to perform any action that the user would be able to perform. A second-order attack, such as persistent XSS, may rely on sending the XSS payload to a back-end database (e.g. through usernames, comments, forum posts, etc.) that gets triggered once a victim loads a webpage containing the relevant information. These attacks are often used to steal sensitive information about a victim such as credit card information, medical records, or cookie details.

### 1.3.3 Distributed denial-of-service

Distributed denial-of-service (DDoS) occurs when a malicious user overwhelms a target resource with superfluous traffic, rendering the resource unable to respond to legitimate traffic in a timely fashion [24]. It should be noted that the superfluous traffic comes from a wide variety of sources (i.e. the "distributed" aspect), which makes it much more difficult to differentiate and block the multiple sources of such traffic. This attack is not specific to web applications, but it remains one of the most common attack patterns due to its generality and effectiveness. The primary purpose of a DDoS attack is to render a web application inoperable, thereby disrupting its normal function and inconveniencing its users. Some secondary purposes that directly result from a DDoS attack include extortion, reputational damage, and financial drain.

## 1.4 Contributions

Many mainstream OS were not designed to handle heavy, data-centric workloads present in many modern applications. DBOS was developed to support these appli-

cations by leveraging a high-performance DBMS in place of the traditional file-based system. Notably, the DBMS supports provenance data capture given its highly structured and accessible design. This prompted the creation of NN, a social media web application that was designed and developed on top of DBOS. It is publically available to allow for real-world traffic and provenance data capture. My main contribution has been the development of a two-part ML model to label the anomalous NN provenance data. The model consists of a tokenizer and convolutional neural network that achieves high performance across a variety of data sources, not just the NN provenance data. A Python daemon was also developed to label NN provenance data in real-time after the initial batch of training and testing. Importantly, this model is an exciting step forward when it comes to developing real-time security analysis features on top of DBOS systems and applications.

# Chapter 2

# Related Work

## 2.1 Web application security

The increased usage and importance of web applications in recent years has prompted focused efforts to bolster web application security and prevent malicious attacks [21, 25, 26, 27, 28, 29]. Intrusion detection systems (IDS) were first created as a general-purpose means to monitor computer systems for suspicious behaviors. These systems operate on the concept of anomaly detection, in which violations of a predefined, normal model constitute deviations of interest. Notably, many of these systems have been adapted in recent years to provide support for web applications [27]. This is done through an extension of the underlying rules-based model employed by most IDS. As the name suggests, a rules-based model relies on a large set of predefined rules to determine anomalous behaviors. Another approach that has gained popularity in recent years is the ML approach. Web applications have become extremely popular in recent years, resulting in large amounts of traffic and logs. This sheer volume of available data is conducive for developing ML models. Both traditional and deep learning schemes have been employed, mostly within the research space to determine malicious web application behaviors [28, 29]. It should be noted that there are two distinct data sources: real-world and generated traffic. Real-world traffic is largely unlabeled and suitable for unsupervised (fully unlabeled) or semi-supervised (partially labeled) models. Generated traffic is always labeled and suitable for supervised

models.

## 2.2 Rules-based models

The traditional approach to detecting malicious web application activity involves a rules-based model [27]. This model is dependent on a large, predefined rule set that is representative of current threats and attacks. As such, the rule set is constantly and continuously updated as new vulnerabilities are discovered. Rules may also be pruned after a designated expiration time to prevent the rule set from becoming too large and unwieldy. The system itself will find and log all rule violations, but the severity dictates whether a violation simply prompts an alert or halts all activity for further analyses.

### 2.2.1 Snort

Snort was originally designed as a packet sniffer and logger developed by Martin Roesch in 1999 [27]. It was a novel system in that it provided cross-platform capabilities and boasted a lightweight deployment, making it readily accessible to many different systems and use cases. Today, it remains one of the most popular IDS given its modular design and open-source code. The system itself consists of four primary subsystems:

- **The packet decoder**. The packet decoder sets pointers within the packet data to facilitate tagging in the detection engine.
- **The preprocessor**. The preprocessor operates between the packet decoder and detection engine to support custom user code. This lets the user modify or analyze the decoded packets before sending them into the detection engine for rule violations. Snort offers various prebuilt preprocessors that are documented in its user manual [30].
- **The detection engine**. The detection engine maintains its detection rules in a two dimensional linked list that consist of Chain Headers and Chain Options.

- The Chain Headers contain common attributes amongst detection rules such as source or destination IP addresses.
- The Chain Options contain modifier options that define the specific signatures of detection rules.

During run-time, the detection engine scans each packet in the forward and reverse directions for any hits, or rule violations. When a hit occurs, the action specified in the rule definition fires, typically triggering the logging/alerting subsystem.

- **The logging/alerting subsystem**. The logging/alerting subsystem consists of separate logging and alerting protocols. The logging protocol can be specified to dump packets in either a decoded, human readable format or tcpdump binary format. The alerting protocol can be called with either the full (alert message and packet header) or fast (alert message and condensed packet header) option.

```
alert tcp any any -> 192.168.1.0/24 111 \
    (content:"|00 01 86 a5|"; msg:"mountd access";)
```

Figure 2-1: Example Snort rule.

All Snort rules follow a common format as depicted in the sample rule from Figure 2-1. The first field specifies whether the packet should be logged, alerted, ignored, blocked, or some combination of the previous options. The second field is the protocol, which can be one of TCP, UDP, ICMP, or IP. The third and fourth fields are the source IP address and port number respectively. The fifth field is the direction operator to indicate whether traffic is flowing in one direction (->) or bidirectionally (<>). The fifth and sixth fields are the destination IP address and port number respectively. The seventh field contains the rule options, which are fully defined in the Snort user manual [30]. Collectively, the sample rule from Figure 2-1 raises an alert when any IP address sends traffic to 192.168.1.0/24:111 that contains "00 01 86 a5" bytecode in the decoded packet. The generated alert will contain the message "mountd access".

## 2.3 Machine learning-based models

Increasing efforts have been focused on developing ML-based models to address web application security [28, 29]. One key advantage of the ML-based approach as opposed to the rules-based approach is the adaptability of ML models. The rules-based approach relies on strict, static rules to detect previously encountered attacks. However, a novel attack can be extremely detrimental in terms of monetary and information losses before it can be successfully documented and incorporated into a rules-based model. An ML model attempts to capture higher-level representations and underlying structure present in the raw data, potentially allowing it to detect novel attacks preemptively. As previously mentioned, ML-based models have only recently become a possibility due to the large volume of web application traffic and logs that have been generated over the past few years. It should be noted that ML techniques can be broadly split into traditional and deep learning approaches. In the following two subsections, I will present a case study from each approach. I will then discuss HTTP2vec [31], an anomaly detection pipeline that uses the same tokenization as this work.

### 2.3.1 Traditional approach case study

Shar et al. explored two traditional ML techniques, logistic regression (LR) and random forest (RF), in both supervised and semi-supervised models for the prediction of web application attacks [28]. The authors noted 32 attributes characterizing input validation and sanitization patterns that are strong indicators of web application vulnerabilities. Broadly, the attributes could be grouped into input accesses, database keywords/operators, delimiters, script tags, and meta-characters. Input accesses focus on the HTTP request parameters, file requests, and database responses. The database keywords/operators determine characters that may have some special meaning to a database query parser. Common delimiters include separators for standard strings (e.g. "), programming language comments (e.g. /*), and nontraditional constructs (e.g. #). Script tags are simply any special characters (e.g. <script>) that have

meaning to a standard HTML interpreter. Meta-characters are characters that have some special meaning within the context of the URL being accessed.

The experiments were performed on seven0 real-world PHP web applications, each with known, exploitable vulnerabilities. The authors utilized the PhpMiner tool from their previous work to derive 15 datasets from the seven web applications based on the available vulnerabilities [32]. The datasets were then manually labeled by the authors for training the proposed models. RF outperformed LR in a supervised model, with a recall of 0.77 and false positive rate of 0.05 averaged over all the datasets. CoForest [33], which extends the co-training paradigm to RF, achieved similar results in a semi-supervised model, with a recall of 0.71 and false positive rate of 0.05 average over all the datasets. These relatively low numbers on recall indicate poor performance from both models.

### 2.3.2 Deep learning approach case study

Pan et al. employed a deep learning approach to web application attack detection by utilizing the Robust Software Modeling Tool (RSMT), which monitors and characterize web application behavior at run-time [29]. Importantly, the authors developed an unsupervised model using a stacked, denoising autoencoder. The RSMT simulated a large number of normal user requests to be fed into the autoencoder. The autoencoder then compressed these requests, retaining key features necessary to reconstruct requests with minimal reconstruction error. A threshold was then set to a value that is greater than or equal to the highest reconstruction error recorded during training. New test requests were considered anomalous if the reconstruction error exceeded that of the threshold. The authors note that the unsupervised model can be extended to a semi-supervised model by manually constructing a small number of anomalous requests. Both normal and anomalous requests can then be run through the autoencoder to generate their respective average reconstruction errors. A threshold is then chosen to maximize some metric, such as $F_1$ score. The authors evaluated their unsupervised approach using RSMT against two sample applications: video management and compression. For the video management application, the autoen-

coder had a precision of 0.898 and recall of 0.942. For the compression application, the autoencoder had a precision of 0.906 and recall of 0.928. Both precision and recall are relatively high across both applications, indicating strong performance from the autoencoder.

### 2.3.3 HTTP2vec

HTTP2vec is a NLP-based anomaly detection pipeline that utilizes RoBERTa [34] to embed HTTP requests for downstream anomaly detection [31]. The HTTP2vec pipeline consists of three main components:

- **Tokenization**. HTTP2vec uses Byte-level Byte Pair Encoding (BBPE) [35] as its tokenizer. Byte Pair Encoding (BPE) is the process of replacing pairs of bytes with bytes that do not exist within the original data, which are referred to as tokens [36]. This allows BPE to serve as a general compression method. BPE has been extended to NLP tasks such that groups of characters are replaced with tokens, with replacements recorded in a dictionary-like object. BBPE extends BPE for NLP tasks in that groups of bytes rather than characters are replaced with tokens. The output from BBPE is fed directly into RoBERTa. Tokenization is an important step in the process since it splits raw text into tokens that can be used to derive semantic meaning.

- **RoBERTa**. RoBERTa is an extension of Bidirectional Encoder Representations from Transformers (BERT), a language representation model developed by Google [37]. The main purpose of BERT is to pretrain non-directional representations of unlabeled text. RoBERTa improves upon BERT through the implementation of dynamic masking, large mini-batches, and larger BBPE. After training, a vector representation of each original HTTP message is derived from the concatenation of the last four RoBERTa layers. This vector representation is then fed into the classification step.

- **Classification**. The authors note that any traditional classification algorithm

is suitable for this step. In particular, logistic regression, random forest, and support vector classification were chosen for downstream anomaly detection.

The authors tested the HTTP2vec pipeline against three datasets: CSIC2010 [38], CSE-CIC-IDS2018 [39], and UMP. Both CSIC2010 and CSE-CIC-IDS2018 were automatically generated using parameter values drawn from known normal and anomalous databases. UMP was was automatically generated using Arachni, a web application security scanner framework [40]. All three classification algorithms performed similarly on each dataset, with overall worst performance on the UMP dataset. In this context, worst performance is defined as AUROC of 0.96 and $F_1$ score of 0.926.

## 2.4 Proprietary solutions

There are many proprietary solutions focused on general-purpose anomaly detection. Some notable examples include Cloudflare, Crowdstrike, and Splunk. As these are proprietary solutions, the exact methods employed by each company are unavailable. However, it is reasonable to assume they use some hybrid of the rules- and ML-based approaches.

# Chapter 3

# Methods

The overall goal of this project was to develop a ML model to detect malicious activity present in the provenance data captured from public accesses to NN. There were three separate sources of labeled HTTP log data for this project, thereby allowing for wide coverage of input vocabulary and attack patterns. The first dataset is publically available from Consejo Superior de Investigaciones Científicas (CSIC), otherwise known as the Spanish National Research Council [38]. The second dataset came from the Cloudflare logs of a software company, Sigma Computing. The third dataset was extracted from the NN provenance capture and labeled using Snort, which is discussed in further detail below [27]. The model consisted of two components: tokenization and classification. During tokenization, BBPE was employed to tokenize all potential input data at the byte-level to be fed into the classification component. During classification, a convolutional neural network (CNN) was trained and evaluated on a particular data source using the outputs from tokenization. A Python daemon was also developed to deploy the model in real-time. The daemon periodically polls the Vertica database hosting the NN provenance data for new entries, runs the new entries through the full model, and pushes the outputted (normal or anomalous) labels into a separate Vertica database.

## 3.1 Datasets

### 3.1.1 Public HTTP logs

```
POST http://localhost:8080/tienda1/publico/autenticar.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=D6037A58019A61C4E5745B952029D61F
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 118

modo=entrar&login=bob%40%3CSCRipt%3Ealert%28Paros%29%3C%2FscrIPT%3E.parosproxy.org&pwd=84m3ri156&remember=on&B1=Entrar
```

Figure 3-1: Example anomalous request from CSIC.

The public HTTP logs consisted of data procured from CSIC [38]. The general schema for the data is shown in Figure 3-1 for an example anomalous request. Notably, this dataset provided a good balance of normal and anomalous traffic, with 36,000 normal and 25,000 anomalous requests. The anomalous requests also cover a wide range of known attack archetypes including SQL injection, buffer overflow, information gathering, and files disclosure.

### 3.1.2 Sigma Computing Cloudflare data

This dataset was procured from Cloudflare annotations of a real-world application deployed by Sigma Computing. For confidentiality purposes, meta information regarding the dataset cannot be disclosed.

### 3.1.3 Nectar Network provenance capture

As previously mentioned, the overall goal of this project was to develop real-time anomaly detection for NN as a proof-of-concept for future endeavors to build security features for DBOS. An example of the NN provenance capture data schema can be seen in Figure 1-3. The final dataset considered for training and testing purposes

consisted of 26,532 HTTP log records, which were generated over a three month period.

These logs were labeled using the "Registered" ruleset for Snort v2.9.19 [27]. Snort rulesets often contain commented out rules, which may result from them being outdated, domain-specific, or overcomplicated. During our deployment of Snort for labeling, we utilized all available rules, totaling over 43,091. This conscious decision was motivated by the small amount of HTTP log records (∼26,000) that needed to be labeled and the ability to confirm the limited rule violations manually afterwards. Each HTTP log record with any number of rule violations was considered to be anomalous while those with no rule violations were considered to be normal. In total, 24,682 logs were found to be anomalous, and 1,850 logs were found to be normal. This is consistent with the assumption that most web crawlers accessing unlisted websites are malicious in nature.

## 3.2 Model

### 3.2.1 Byte-level Byte Pair Encoding

BPE was introduced as a method of compressing strings [36]. The technique uses the characters of a string as tokens, and additionally adds tokens representing the most common combinations of characters present in a string. By doing so, the author reported that BPE was able to outperform Lempel–Ziv–Welch compression in terms of compressed data size at the cost of increased time for compression. Note that after tokenization, there is a dictionary mapping seen characters to assigned tokens known commonly as a vocabulary.

Because BPE can tokenize a string without loss of information, it can serve as a tokenizer for language machine learning techniques [41]. The authors report that this tokenization method serves well for vocabularies in which there are very rare words and words which are out of vocabulary.

BBPE is a tokenization method which follows the same techniques as BPE, but

operates on bytes instead of characters [35]. This is particularly powerful because BBPE guarantees that there will be no unknown tokens. In the worst case, an input can be tokenized as its individual bytes, meaning unique characters that have not been seen before can still be tokenized. Because HTTP requests and other machine code often includes unique characters, and in particular injection attacks use unique characters to confuse web application, this characteristic makes BBPE a strong choice for tokenizing our inputs.

Incoming HTTP requests were formatted into a single string which included all of the fields separated by spaces. An example string is `"GET http://url.com/path HTTP/1.1 [User-Agent] [Content-Length] ..."`, in which `...` represents additional HTTP fields. These were then collected and used to train the BBPE tokenizer.

BBPE is particularly useful for machine learning because it does not require labeled data to find a good tokenization of the entire dataset. This is relevant to applications involving malicious HTTP requests since it is easier to obtain unlabeled rather than labeled data.

### 3.2.2 Convolutional Neural Network

Once BBPE tokenizes the HTTP request, the request is classified by a CNN model. The full architecture of the model is shown in Figure 3-2.

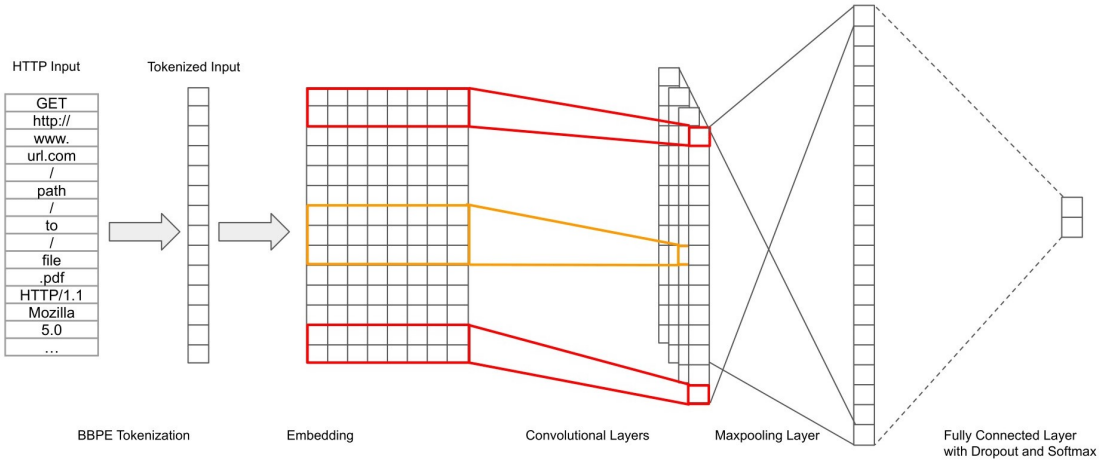



Figure 3-2: Model architecture

- **Embedding tokens**. Zhang et al. shows that learning embedded tokens as part of the CNN works well for task-specific text applications, such as detecting web attacks [42]. This helps the learned embedding relate more closely with the desired classification, in this case whether an HTTP request is malicious. This stands in contrast to embedding techniques such as Word2Vec, which aims to learn the semantic meaning of tokens. While this makes sense for actual languages, HTTP requests often lack this semantic meaning. While some tokens are words, other might simply be characters such as `"%20"`, the URL escape sequence for the space character.

- **Convolution layers**. For the convolutional layers, 3 kernel sizes (2, 3, 4) were used. For each of these kernel sizes, 100 filters were used reaching a total of 300 filters in this layer. Each of these layers uses the ReLU activation function, a piecewise linear function that returns positive values directly and negative values as zeroes. This generates the possible feature map of the HTTP request.

- **Maxpooling layers**. This layer takes the maximum of each feature map generated by the convolutional layers and concatenates them into a single vector. This selects which features are important to the classifier and reduces the number of inputs into the neural network.

- **Dense layer**. The final layer densely connects the concatenated maxpool layers to our 2 classes, malicious or benign. In addition, to avoid overfitting during training, a dropout layer was included which zeroed out inputs with a probability of $p = 0.2$. A softmax layer is added to make the decision on the final label of an input.

## 3.3 Python daemon

A Python daemon can theoretically be deployed by running a standard Python script from the command line with an ampersand (&). This would send the process to the background. However, this simplistic solution is not well-behaved for several reasons.

Most notably, it fails to handle process cleanup on interrupt and maintain a process ID (PID) file while running. This is resolved through usage of the daemon library, which implements the well-behaved daemon specification outlined by Python Enhancement Proposals (PEP) 3143.

The Python daemon is relatively straightforward with respect to its operation. Upon startup, the pretrained model is loaded into the Python environment, and the connection to the Vertica database housing the NN provenance data is established. While the daemon is running, a variable maintains the highest epoch encountered after each Vertica table poll. These polls occur every 5 seconds, with the highest epoch variable ensuring that only new, unlabeled data is being fed into the daemon workload. Once new data is polled, which signifies a new highest epoch and updates the highest epoch variable, the daemon tokenizes the input and feeds it into the pretrained model for classification. The labeled data is then pushed onto a separate Vertica table with the same format as the original Vertica table except for an additional float column with the predicted probability of anomaly.

# Chapter 4

# Results

## 4.1 Evaluation metrics

In order to evaluate the model, the first natural metric is accuracy, the proportion of events which are correctly labeled. However, this isn't fully illustrative. For example, in the Nectar Network dataset a high proportion of events captured are malicious in nature. As such, a classifier could label all events as malicious and still achieve a high accuracy due to the low total number of non-malicious events. As additional metrics, we will also report the precision, recall, and $F_1$ score for our datasets.

$$\begin{aligned} \text{Precision} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \\ \text{Recall} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \\ F_1 \text{ score} &= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned}$$

Precision serves to guarantee that the false positive rate is low, while recall guarantees that the false negative rate is low. The $F_1$ score serves as a way to represent both the precision and recall using their harmonic mean. For all of these metrics, the score for a perfect classifier would be 1.

These metrics ensure that a classifier is performing well despite a very unbalanced dataset. Regardless of which class is underrepresented in the dataset, either precision or recall can capture the fact that a classifier is misrepresenting that class. If either is low, the $F_1$ score will likewise suffer.

## 4.2 Public HTTP logs

Table 4.1 shows the performance of the model when run with 80% of the data used for training.

Table 4.1: Performance on CSIC public dataset

| Metric | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Value | 0.99821 | 0.99774 | 0.99788 | 0.99781 |

Additionally, because this dataset is publically available, comparisons with other published techniques are feasible. As shown in Table 4.2, BBPE CNN outperforms all other published techniques, despite very close resemblance to HTTP2vec. While HTTP2vec also uses BBPE to tokenize the inputs, it then learns the vector embedding via RoBERTa and classifies the vector using traditional machine learning methods. The results shown below are using Support Vector Classification, and are the best reported on the CSIC dataset. It seems that the combination of BBPE and CNN is what make this technique so powerful.

Table 4.2: Comparison on CSIC Data

| Method | F1 Score |
|---|---|
| BBPE CNN | 0.998 |
| HTTP2vec [31] | 0.969 |
| Code Level CNN [43] | 0.963 |
| SAE [44] | 0.841 |

Table 4.3 list how the performance of the model changes with different amounts of training data. While performance does decrease, it's surprising just how well the model can do with only 5% (∼3,000) examples to train on. This suggests that the

BBPE, which is trained on the entire dataset, is helping to clarify the important features in our data and making it easier for the CNN to learn a proper classifier.

One possible instance of this lies in SQL injection attacks, which often obscure their intent by using URL encoding. `%25` decodes as `%`, which is the URL escape character. This can be exploited against poorly secured web applications. Since BBPE operates at the byte level, it is able to recognize `%25` as unique token, which gives the CNN a very easy way to identify this as a feature of attacks.

Table 4.3: Performance and train-test split

| % Train Data | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 80% | 0.99821 | 0.99774 | 0.99788 | 0.99781 |
| 50% | 0.99575 | 0.99443 | 0.99515 | 0.99479 |
| 25% | 0.98541 | 0.96998 | 0.99486 | 0.98226 |
| 10% | 0.95726 | 0.90775 | 0.99622 | 0.94993 |
| 5% | 0.95361 | 0.94670 | 0.93887 | 0.94276 |

## 4.3 Sigma Computing Cloudflare data

The CSIC public dataset is automatically generated, which is why it can manage to be fairly balanced. In real world applications, this is rarely the case. The Sigma dataset represents real world activity, in which a vast majority of the traffic logged is benign in nature. Less than 4% of the events logged are malicious, so it is necessary to consider the $F_1$ score in order to accurately understand the model's performance. Notably, all performance metrics are scored from 0 to 1, indicating that the model is a near-perfect classifier for this dataset.

Table 4.4: Performance on Sigma dataset

| Metric | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Value | 0.99962 | 0.99886 | 0.99210 | 0.99660 |

## 4.4 Nectar Network provenance capture

The final dataset came from NN as previously described. Since the website was available for access by anyone, the majority of the traffic it generated was webcrawlers or malicious users. As a result, the data is heavily weighted towards events labeled as malicious, with only about 7% of the traffic being benign. We again consult the $F_1$ score to properly evaluate the model. The model is a near-perfect classifier for this dataset since all performance metrics are almost exactly 1.

Table 4.5: Performance on NN dataset

| Metric | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Value | 0.99909 | 0.99953 | 0.99949 | 0.99951 |

# Chapter 5

# Closing Remarks

## 5.1 Discussion

The results of the ML model across all three datasets are extremely promising, with $F_1$ scores above 0.99. Additionally, the model requires minimal training data to achieve such a high $F_1$ score as demonstrated in Table 4.3. Despite the success of this ML model, there are several opportunities for further exploration that would bolster the overall purpose of the project. Recall that the main goal of this work is to demonstrate a robust ML model running in real-time against provenance data captured by DBOS. First, the datasets all came from different sources, but they were all ultimately labeled using an existing rules-based model. Due to unforeseen circumstances, I was unable to attain manually labeled data that would serve as more reasonable ground truth. As such, the results of the ML model, while impressive, only represent its ability to match the results achieved by a rules-based model. Second, the Python daemon constitutes the real-time aspect of the overall project. Further investigation should be dedicated to stress testing the Python daemon under large traffic loads. At the height of NN activity, there were only 2,203 requests over the course of a minute. For reference, one of the largest social media platforms in the world, Facebook, receives over 780 million queries per minute [45]. This number was reported over a decade ago in 2010, and has most certainly increased dramatically since then. While we don't expect to ever match the level of traffic that Facebook receives, it is necessary

to test the scalability and performance of the current approach and explore potential optimizations if the need arises.

### 5.1.1 Semi-supervised ML model

The current model relies solely on labeling provided by existing rules-based models. An interesting future direction would be to train a semi-supervised ML model based on a small amount of manually labeled data. Recall from the "Related Work" chapter that Pan et al. achieved high performance using such an approach with an autoencoder [29]. Most deep learning problems require a large amount of data to produce significant results, but the success of the CNN developed on ∼25,000 entries suggests that a semi-supervised approach may be feasible with a relatively small amount of labeled data. This is further supported by the results in Table 4.3, in which the model achieved an admirable $F_1$ score of 0.94 on slightly more than 1,000 labeled entries. Such a model would be more representative of the anomaly detection process that would be adopted by a security expert or administrator.

### 5.1.2 Python daemon performance testing

The Python daemon currently processes the NN provenance data in real-time based on incoming traffic. However, NN is a small-scale social media web application with very basic functionality. It lacks the level of traffic that would be expected of an application deployed by a large- or even medium-sized organization. It then becomes necessary to test the scalability and performance of the Python daemon manually. This would allow me to determine potential bottlenecks and identify optimizations in the Python daemon code. At the same time, this would provide further reassurance that reasonably-sized applications can be deployed on a DBOS system, with a robust real-time anomaly detection system.

## 5.2 Conclusion

Overall, this work has been a promising starting point for real-time security features developed on top of a DBOS application. The deployed model achieves extremely high performance across all evaluation metrics, with relatively minimal required training data. Concretely, I evaluated the ML model across three separate data sources that were generated from both academic and industry settings. Across all three datasets, the $F_1$ score, a metric that is dependent on both precision and recall, never dipped below 0.99. Taking a step back, more and more modern tasks have become dependent on data-centric workloads. Importantly, the data contained within these workloads are often sensitive or personal in nature, requiring a strong security system to prevent unauthorized accesses and/or modifications. Historically, data breaches and leaks have had huge socioeconomic impacts all around the world. As DBOS attempts to address the needs of data-centric workloads, this project takes an active first step in addressing the needs of an underlying, strong, real-time security analytics model for DBOS.

# Bibliography


[1] A. Skiadopoulos, Q. Li, P. Kraft, K. Kaffes, D. Hong, S. Mathew, D. Bestor, M. J. Cafarella, V. Gadepally, G. Graefe, J. Kepner, C. Kozyrakis, T. Kraska, M. Stonebraker, L. Suresh, and M. Zaharia, "DBOS: A dbms-oriented operating system," *Proc. VLDB Endow.*, vol. 15, no. 1, pp. 21–30, 2021.

[2] D. A. Patterson, "Past and future of hardware and architecture," in *SOSP History Day 2015, Monterey, California, USA, October 4, 2015*, pp. 9:1–9:63, ACM, 2015.

[3] D. Ritchie and K. Thompson, "The UNIX time-sharing system (abstract)," in *Proceedings of the Fourth Symposium on Operating System Principles, SOSP 1973, Thomas J. Watson, Research Center, Yorktown Heights, New York, USA, October 15-17, 1973* (H. Schorr, A. J. Perlis, P. Weiner, and W. D. Frazer, eds.), p. 27, ACM, 1973.

[4] S. H. Bokhari, "The linux operating system," *Computer*, vol. 28, no. 8, pp. 74–79, 1995.

[5] G. Shainer, R. L. Graham, C. J. Newburn, O. R. Hernandez, G. Bloch, T. Gibbs, and J. C. Wells, "Nvidia's cloud native supercomputing," in *Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation - 21st Smoky Mountains Computational Sciences and Engineering, SMC 2021, Virtual Event, October 18-20, 2021, Revised Selected Papers* (J. Nichols, A. B. Maccabe, J. J. Nutaro, S. Pophale, P. Devineni, T. Ahearn, and B. Verastegui, eds.), vol. 1512 of *Communications in Computer and Information Science*, pp. 340–357, Springer, 2021.

[6] O. Challabi, R. Zenki, and M. O. Agyeman, "A study of fpga-based supercomputing platforms," in *ISCSIC 2019: 3rd International Symposium on Computer Science and Intelligent Control, Amsterdam, The Netherlands, September 25-27, 2019*, pp. 52:1–52:5, ACM, 2019.

[7] A. Prout, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, M. Houle, M. Jones, P. Michaleas, L. Milechin, J. Mullen, A. Rosa, S. Samsi, A. Reuther, and J. Kepner, "MIT supercloud portal workspace: Enabling HPC web application deployment," in *2017 IEEE High Performance Extreme Computing Conference, HPEC 2017, Waltham, MA, USA, September 12-14, 2017*, pp. 1–6, IEEE, 2017.

[8] C. Byun, A. Klein, L. Milechin, P. Michaleas, J. Mullen, A. Prout, A. Rosa, S. Samsi, C. Yee, A. Reuther, J. Kepner, W. Arcand, D. Bestor, W. Bergeron, M. Hubbell, V. Gadepally, M. Houle, and M. Jones, "Optimizing xeon phi for interactive data analysis," in *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019, Waltham, MA, USA, September 24-26, 2019*, pp. 1–6, IEEE, 2019.

[9] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *2020 IEEE High Performance Extreme Computing Conference, HPEC 2020, Waltham, MA, USA, September 22-24, 2020*, pp. 1–12, IEEE, 2020.

[10] A. Reuther, C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, M. Jones, P. Michaleas, A. Prout, A. Rosa, and J. Kepner, "Scalable system scheduling for HPC and big data," *J. Parallel Distributed Comput.*, vol. 111, pp. 76–92, 2018.

[11] N. Jouppi, "Quantifying the performance of the tpu, our first machine learning chip," 2017.

[12] A. Orel and I. Bernik, "GDPR and health personal data; tricks and traps of compliance," in *Decision Support Systems and Education - Help and Support in Healthcare, Special Topic Conference of the European Federation for Medical Informatics, EFMI-STC 2018, Zagreb, Croatia, 15-16 October 2018* (J. Mantas, Z. Sonicki, M. Crisan-Vida, K. Fister, M. Hägglund, A. Kolokathi, and M. Hercigonja-Szekeres, eds.), vol. 255 of *Studies in Health Technology and Informatics*, pp. 155–159, IOS Press, 2018.

[13] J. Zerlang, "GDPR: a milestone in convergence for cyber-security and compliance," *Netw. Secur.*, vol. 2017, no. 6, pp. 8–11, 2017.

[14] K. Djemame, M. Parker, and D. Datsev, "Open-source serverless architectures: an evaluation of apache openwhisk," in *13th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2020, Leicester, United Kingdom, December 7-10, 2020*, pp. 329–335, IEEE, 2020.

[15] Z. Jia and E. Witchel, "Boki: Stateful serverless computing with shared logs," in *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021* (R. van Renesse and N. Zeldovich, eds.), pp. 691–707, ACM, 2021.

[16] D. Kumar, Q. Li, J. Li, P. Kraft, A. Skiadopoulos, L. Suresh, M. J. Cafarella, and M. Stonebraker, "Data governance in a database operating system (DBOS)," in *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB Workshops, Poly 2021 and DMAH 2021, Virtual Event, August 20, 2021, Revised Selected Papers* (E. K. Rezig, V. Gadepally, T. G. Mattson, M. Stonebraker, T. Kraska, F. Wang, G. Luo, J. Kong, and A. Dubovitskaya, eds.), vol. 12921 of *Lecture Notes in Computer Science*, pp. 43–59, Springer, 2021.

[17] M. Stonebraker and A. Weisberg, "The voltdb main memory DBMS," *IEEE Data Eng. Bull.*, vol. 36, no. 2, pp. 21–27, 2013.

[18] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear, "The vertica analytic database: C-store 7 years later," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1790–1801, 2012.

[19] S. White, J. Martinez, D. Coulter, D. Batchelor, A. Laforge, M. Jacobs, and M. Satran, "W3c logging," jun 2021. https://docs.microsoft.com/en-us/windows/win32/http/w3c-logging.

[20] J. Conallen, "Modeling web application architectures with UML," *Commun. ACM*, vol. 42, no. 10, pp. 63–70, 1999.

[21] X. Li and Y. Xue, "A survey on server-side approaches to securing web applications," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 54:1–54:29, 2014.

[22] W. Baker, M. Goudie, A. Hutton, C. D. Hylender, J. Niemantsverdriet, C. Novak, D. Ostertag, C. Porter, M. Rosen, B. Sartin, and P. Tippett.

[23] A. Dizdar, "Sql injection attack: Real life attacks and code examples," 2022.

[24] A. Praseed and P. S. Thilagam, "Ddos attacks at the application layer: Challenges and research perspectives for safeguarding web applications," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 1, pp. 661–685, 2019.

[25] S. Mishra, S. K. Sharma, and M. Alowaidi, "Analysis of security issues of cloud-based web applications," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 7, pp. 7051–7062, 2021.

[26] H. Huang, Z. Zhang, H. Cheng, and S. W. Shieh, "Web application security: Threats, countermeasures, and pitfalls," *Computer*, vol. 50, no. 6, pp. 81–85, 2017.

[27] M. Roesch, "Snort: Lightweight intrusion detection for networks," in *Proceedings of the 13th Conference on Systems Administration (LISA-99), Seattle, WA, USA, November 7-12, 1999* (D. W. Parter, ed.), pp. 229–238, USENIX, 1999.

[28] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning," *IEEE Trans. Dependable Secur. Comput.*, vol. 12, no. 6, pp. 688–707, 2015.

[29] Y. Pan, F. Sun, Z. Teng, J. White, D. C. Schmidt, J. Staples, and L. Krause, "Detecting web attacks with end-to-end deep learning," *J. Internet Serv. Appl.*, vol. 10, no. 1, pp. 16:1–16:22, 2019.

[30] S. Team, "SNORT users manual," 2016.

[31] M. Gniewkowski, H. Maciejewski, T. R. Surmacz, and W. Walentynowicz, "Http2vec: Embedding of HTTP requests for detection of anomalous traffic," *CoRR*, vol. abs/2108.01763, 2021.

[32] L. K. Shar, H. B. K. Tan, and L. C. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013* (D. Notkin, B. H. C. Cheng, and K. Pohl, eds.), pp. 642–651, IEEE Computer Society, 2013.

[33] M. Li and Z. Zhou, "Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples," *IEEE Trans. Syst. Man Cybern. Part A*, vol. 37, no. 6, pp. 1088–1098, 2007.

[34] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.

[35] C. Wang, K. Cho, and J. Gu, "Neural machine translation with byte-level subwords," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 9154–9160, AAAI Press, 2020.

[36] P. Gage, "A new algorithm for data compression," The C User Journal, 1994.

[37] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), pp. 4171–4186, Association for Computational Linguistics, 2019.

[38] C. T. Giménez, A. P. Villegas, and G. Álvarez Marañón, "Http dataset csic 2010," 2010.

[39] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018* (P. Mori, S. Furnell, and O. Camp, eds.), pp. 108–116, SciTePress, 2018.

[40] Tasos Laskos, "Arachni."

[41] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, The Association for Computer Linguistics, 2016.

[42] M. Zhang, B. Xu, S. Bai, S. Lu, and Z. Lin, "A deep learning method to detect web attacks using a specially designed CNN," in *Neural Information Processing - 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part V* (D. Liu, S. Xie, Y. Li, D. Zhao, and E. M. El-Alfy, eds.), vol. 10638 of *Lecture Notes in Computer Science*, pp. 828–836, Springer, 2017.

[43] I. Jemal, M. A. Haddar, O. Cheikhrouhou, and A. Mahfoudhi, "Malicious http request detection using code-level convolutional neural network," in *Risks and Security of Internet and Systems - 15th International Conference, CRiSIS 2020, Paris, France, November 4-6, 2020, Revised Selected Papers* (J. García-Alfaro, J. Leneutre, N. Cuppens, and R. Yaich, eds.), vol. 12528 of *Lecture Notes in Computer Science*, pp. 317–324, Springer, 2020.

[44] A. M. Vartouni, S. S. Kashi, and M. Teshnehlab, "An anomaly detection method to detect web attacks using stacked auto-encoder," in *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, pp. 131–134, 2018.

[45] M. Callahan, "Running mysql at scale tech talk." https://www.facebook.com/watch/?v=695491248045, Nov 2010.