ENHANCING CLOUD SECURITY AND PRIVACY WITH ZERO-KNOWLEDGE ENCRYPTION AND VULNERABILITY ASSESSMENT IN KUBERNETES DEPLOYMENTS

By

Ali Alqarni

APPROVED:

Graduate Committee:

_____

Supervisor Dr. Yi Gu (Computer Science)

_____

Dr. Joshua L. Phillips (Computer Science)

_____

Dr. Arpan Sainju (Computer Science)

_____

Dr. Medha Sarkar, Chairperson Computer Science Department

_____

Dr. David Butler, Dean of the College of Graduate Studies

Enhancing Cloud Security and Privacy with Zero-Knowledge Encryption and Vulnerability Assessment in Kubernetes Deployments

By

Ali Alqarni

A thesis submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

Middle Tennessee State University

(May 2023)

ACKNOWLEDGEMENTS

I would like to thank MTSU and Computer Science department for everything. I am indebted to my supervisor, Prof. Yi Gu, for her continued guidance and endless supply of fascinating projects. I would also like to thank Prof. Joshua L. Phillips and Prof. Arpan Sainju. Words cannot express my gratitude to my parents, Khowilid and Nourh. Your unwavering support and belief in me have helped me get to this day. Finally, I am deeply grateful to my siblings Jawharah, Khaled, Sarah, Haya, Muhammad, and my wife, Amjad, for your sacrifices and support throughout my lifelong educational pursuits, especially over the last years of this process.

ABSTRACT

Cloud computing has become increasingly significant in recent years, yet security concerns and the challenges of rapidly building, testing, and deploying systems in monolithic environments can hinder innovation. Kubernetes provides a practical approach for distributed systems. In this thesis, we investigate an integrated framework to enhance the security and privacy of a Django-based application deployed on an open-source Kubernetes cluster, adding hybrid encryption and zero-knowledge encryption, and identifying vulnerabilities. This work contributes to understanding how Django's built-in security features can be effectively combined with Kubernetes deployment to provide a potentially robust web application environment. The results demonstrate that Django and Kubernetes can be effectively combined to create an efficient application deployment platform showing minimal vulnerabilities using monitoring tools such as Kube-hunter, Datree, and Mozilla Observatory. Also, our results showcase a practical implementation of zero-knowledge encryption and how it can be applied in a real-world setting.

TABLE OF CONTENTS

LIST OF TABLES vi

LIST OF FIGURES vii

LIST OF SYMBOLS AND ABBREVIATIONS ix

Chapter

I. INTRODUCTION 1

Contributions 3

II. BACKGROUND 4

CIA triad: Availability, Integrity, and Confidentiality 4

Cloud service models 5

Vulnerabilities 6

Monitoring tools 8

III. RELATED WORK 9

Django-based applications 9

Kubernetes-Based Environments 10

Hybrid Encryption Schemes 10

Zero-Knowledge Encryption 11

Overcoming the Gap in Related Work 11

IV. PROBLEMS AND METHODS 12

V. RESULTS 29

VI. DISCUSSION 45

VII. FUTURE WORK 46

BIBLIOGRAPHY 47

APPENDIX 51

LIST OF TABLES

Table 1 – An example of how to insert a file into the database. 29

Table 2 – Results of encryption methods for stored files 32

Table 3 – IP address blocking due to excessive requests 38

Table 4 – Comparison between Kubernetes and standard servers 44

LIST OF FIGURES

Figure 1 – The top 10 web application vulnerabilities 6

Figure 2 – The database security information with VPC 12

Figure 3 – The diagram showing Docker as a container for the application images ... 13

Figure 4 – The complete diagram that shows when the application connects with nodes14

Figure 5 – Configuration deployment yaml file. 15

Figure 6 – Configuration service yaml file. 16

Figure 7 – Configuration ingress yaml file. 17

Figure 8 – Configuration Role, RoleBinding, and ServiceAccount yaml files. 18

Figure 9 – Nodes of the application 19

Figure 10 – The figure showing the connection between database storage and nodes 20

Figure 11 – This figure shows the user can connect with any node..... 21

Figure 12 – This diagram shows the first step of uploading a file to the server by user..23

Figure 13 – The registration Page 24

Figure 14 – The login Page 25

Figure 15 – This diagram shows that any node can help for uploading file 25

Figure 16 – The three options for encryption of a file when uploading..... 26

Figure 17 – This diagram shows a node helps to upload the file 26

Figure 18 – This diagram shows the file is stored without encryption 27

Figure 19 – This diagram shows the file is stored with encryption from server-side .. 27

Figure 20 – This diagram shows the file is stored with encryption from the client-side28

Figure 21 – This figure shows an example of how to insert a file into the database.... 30

Figure 22 – Zero Knowledge Encryption 31

Figure 23 – The results of how the file encrypted with the three options of encryptions on List Upload Interface 34

Figure 24 – Monitoring from Datree website A 34

Figure 25 – Monitoring from Datree website B 35

Figure 26 – Monitoring from Kube-hunter website A 35

Figure 27 – Monitoring from Kube-hunter website B 36

Figure 28 – Monitoring from Mozilla Observatory website A 37

Figure 29 – Monitoring from Mozilla Observatory website B 38

Figure 30 – The login page with a warning message. 39

LIST OF SYMBOLS AND ABBREVIATIONS

SaaS - Software as a Service

PaaS - Platform as a Service

IaaS - Infrastructure as a Service

AES - Advanced Encryption Standard

VPN - Virtual Private Network

AWS - Amazon Web Services

APIs - Application Programming Interfaces

MTSU - Middle Tennessee State University

ORAM - Oblivious Random Access Machine

OWASP - Open Web Application Security Project

XSS - Cross-Site Scripting

NIDS - Network Intrusion Detection System

IPS - Intrusion Prevention System

ZAP - Zed Attack Proxy

FP - False Positives

FN - False Negatives

ORM - Object-Relational Mapping

IaC - Infrastructure as Code

VPC - Virtual Private Cloud

CDN - Content Delivery Networks

RBAC - Role-Based Access Control

ZKE - Zero Knowledge Encryption

DRY - Don't Repeat Yourself

PVs - Persistent Volumes

PVCs - Persistent Volume Claims

CHAPTER I

INTRODUCTION

Over time, physical servers transitioned into providers of hardware resources, with virtual servers created on the physical servers using virtualization tools. This approach, known as virtualization, allowed for deploying applications on virtual servers. Subsequently, virtualization advanced to containerization, whereby applications were deployed within containers. Finally, cloud computing emerged as technology evolved, transforming how businesses and individuals accessed and managed computing resources [10].

In recent years, there has been a significant increase in the utilization of the Internet and cloud services [3] and web applications [29]. Cloud computing enabled the provision of on-demand, scalable infrastructure, and services, allowing users to access resources over the internet without the need to maintain and manage their hardware. Also, it facilitates on-demand access to computing resources such as data storage and processing power without necessitating direct active management by the user. The onset of the COVID-19 pandemic in 2020 accelerated the adoption of cloud services across numerous sectors, including work, research, conferences, and corporate applications [3,30]. Also, with the rise of the ubiquitous use of web applications, the demand for web application development has thrived [29].

Despite the rapid growth in cloud-based interest and web applications, there are various concerns, such as data privacy and security [3,29]. Data privacy in cloud computing involves the secure collection, storage, transfer, and sharing of information without jeopardizing individuals' privacy [22]. However, malicious individuals, such as hackers,

may exploit this data; hence, cloud professionals must continuously improve privacy measures. Therefore, numerous studies focus on strengthening cloud infrastructure [16].

Vulnerabilities in web applications occur due to a lack of knowledge of the web developers or due to built-in flaws in the platform used. Inexperienced web developers who may not be fully aware of secure code practices end up designing applications with high-security risks. When launched without prior security testing, these apps become an attractive target for attackers. Any user who uses such vulnerable applications becomes prey for the hacker and their privacy and confidential information being compromised [31, 32].

One strategy to enhance privacy in cloud computing and safeguard user data is encryption. Encryption entails encoding information by transforming the original representation, or plaintext, into an alternative form called ciphertext. Ideally, only authorized parties can decode a ciphertext or image back to plaintext and access the original information [33]. Building on the concept of encryption, certain web app development frameworks, like Django, provide robust security features and practices to enhance privacy and data protection further [37].

Django has many built-in security features, including protection against cross-site scripting and cross-site request forgery. It also has a secure password hashing algorithm and a built-in user authentication system that can help prevent unauthorized access [34]. However, managing their deployment and infrastructure becomes increasingly important as web applications grow in complexity and scale. This is where Kubernetes [36], an open-source container orchestration platform, enters the picture. Kubernetes clusters provide a powerful and flexible platform for deploying and managing containerized applications at

scale. Also, it can help improve security for containerized applications by providing strong access controls and network security policies [35].

Contributions

We answer the following research question: **RQ: How effective is the combination of Django and Kubernetes, and integrating hybrid encryption, zero-knowledge encryption, and attack prevention measures?**

Our contributions are as follows:

1. We effectively combine Django and Kubernetes, leveraging their strengths while integrating hybrid encryption, zero-knowledge encryption, and attack prevention measures.
2. We investigate the potential synergies between Django, Kubernetes, hybrid encryption, and zero-knowledge encryption in enhancing application security and performance and efficiently blocking malicious IP.
3. We evaluate the application's security posture using monitoring tools like Kube-hunter, Datree, and Mozilla Observatory, enabling us to identify and address potential vulnerabilities and maintain a high level of protection for the infrastructure.

CHAPTER II

BACKGROUND

CIA triad: Availability, Integrity, and Confidentiality

The CIA triad, which means availability, integrity, and confidentiality, is a model for making information security rules in a company or organization. It helps ensure the organization's data is safe and easy to get when needed [23, 24].

Confidentiality is crucial in organizations' safeguarding of sensitive data and preserving privacy. Achieving confidentiality necessitates regulating information access to avert unauthorized data sharing, whether intentional or accidental [2]. In addition, preventing unauthorized individuals from accessing valuable business assets is critical to maintaining confidentiality. In Cloud systems, confidentiality centers around protecting users' data. These systems often transmit data through private networks, prone to attacks. Implementing a specialized VPN and firewall can enhance the security of databases on private networks [2]. This approach aims to maintain the confidentiality of data while navigating the vulnerabilities of Cloud systems.

Integrity involves making sure the data is trustworthy and free from tampering. The integrity of the data is maintained only if the data is authentic, accurate, and reliable. Data integrity in the Cloud system means preserving information integrity (i.e., not lost or modified by unauthorized users). Data is the base for Cloud Computing services, such as Data as a Service, Software as a Service, and Platform as a Service, so keeping data integrity is a fundamental task [39]. In addition, cloud computing usually provides massive data processing capability [38].

Even if data is kept confidential and its integrity maintained, it is only useful if available to those in the organization and the customers they serve. This means that systems, networks, and applications must function as they should and when. Also, individuals with access to specific information must be able to consume it when needed and get to the data within the reasonable access time [40].

Cloud service models

The three primary Cloud service models are software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) [5, 7, 21]. Although these models have distinctive differences, they all face similar privacy-related challenges. Cloud providers must enhance privacy in the cloud, raise user awareness of potential issues, and develop customer trust in cloud privacy to effectively use services and share data. Despite the rapid growth in cloud-based interest, various concerns exist, such as data privacy. Data privacy in cloud computing involves the secure collection, storage, transfer, and sharing of information without jeopardizing individuals' privacy [22]. However, malicious individuals, such as hackers, may exploit this data; hence, Cloud professionals must continuously improve privacy measures. In addition, numerous studies focus on strengthening cloud infrastructure. For example, Daniel [16] suggested solutions for the challenges of privacy and reliability in cloud computing. Containerization tools operate as services on virtual or physical servers and manage application containers, thus facilitating the efficient transfer of applications as containers between servers [9]. Numerous enterprises have embraced the Internet as a cost-effective and user-friendly platform for communication, information exchange with potential clients, and conducting transactions with users. With the rise of the ubiquitous use of web applications, the demand for web

application development has thrived [6]. The elegance of web application development is that young developers with exciting ideas can form a group to develop interesting web applications [11].

Vulnerabilities

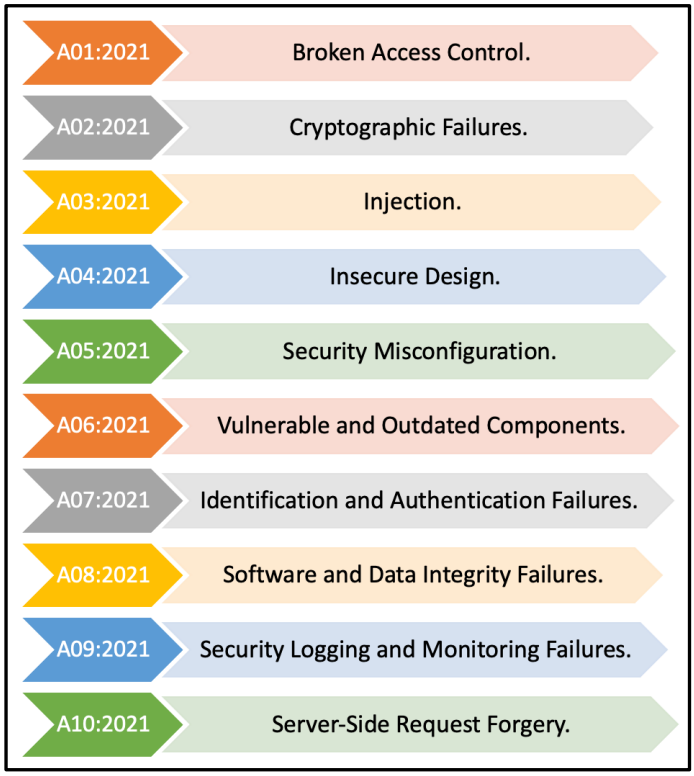


Figure 1. The OWASP top 10 web application vulnerabilities.

In [25] recent years, the Open Web Application Security Project (OWASP) has published a list of the top 10 web application vulnerabilities, which highlights numerous security risks that potentially expose sensitive user data, such as phone numbers, addresses, credit card information, and other valuable information to attackers. These vulnerabilities have implications for both the service provider and the end user. Therefore, it is essential for the provider to continuously update their systems to safeguard the data stored within their infrastructure and protect user data stored in their databases (Figure 1).

Web applications have become integral to our daily lives, providing various services such as e-commerce, online banking, and social networking. However, the rapid development of web technologies has given rise to numerous security vulnerabilities, making web applications attractive targets for malicious attacks. Among the most common and severe threats to web applications are SQL Injection and Cross-Site Scripting (XSS) attacks [4, 40].

As web applications handle sensitive user data, ensuring their security is paramount. However, due to the nature of these applications, the confidentiality of the data and the user's privacy are often compromised [14, 41]. Attackers exploit vulnerable areas, such as login pages or other open access points, to inject malicious scripts that compromise security measures. These attacks must be detected and eliminated before they can impact the integrity and confidentiality of the data.

Several solutions have been proposed to address these issues, including the use of vulnerability scanners to identify and mitigate potential threats. For example, open-source vulnerability scanners, such as OWASP ZAP and Skipfish, have successfully detected SQL Injection and XSS vulnerabilities [17]. However, relying solely on vulnerability scanners can be insufficient due to their limitations in accuracy and coverage.

An alternative approach to detect and prevent SQL Injection and XSS attacks is using the Nginx Reverse Proxy protocols and Suricata NIDS/IPS rules [18]. This method has shown promising results in reducing network overhead and effectively identifying different types of attacks without needing a firewall.

Moreover, incorporating honeypots into Web Application Firewalls (WAF) and utilizing machine learning models has been suggested to improve web application security

[12]. By introducing a honeypot in the network architecture, the application owner can be notified of ongoing attacks while providing a fake response to the attackers, effectively thwarting their attempts.

Monitoring tools

Kube-hunter is a security testing tool for Kubernetes clusters. It is designed to help identify potential vulnerabilities in your Kubernetes environment by actively scanning for known security risks, misconfigurations, and compliance violations. With Kube-hunter, you can get insights into how an attacker might potentially exploit your Kubernetes infrastructure. It can be run remotely to probe public IPs or internally within the cluster to discover and report potential issues [26].

Datree is a policy enforcement and continuous configuration management tool that ensures developers follow best practices when using Kubernetes and other infrastructure as code (IaC) platforms. It helps organizations maintain consistency, avoid errors, and enforce compliance by automatically scanning and validating Kubernetes configurations, Terraform files, and other IaC files against customizable rules. "Datree Feature" refers to a specific feature or functionality within the Datree platform [27].

Mozilla Observatory is a free online service developed by Mozilla to help developers analyze and improve the security of their websites. The tool performs various tests on a given website to evaluate its security configuration, such as checking for HTTPS, Content Security Policy, HTTP Strict Transport Security, and other best practices. After the analysis, the tool provides a report with a security score, a list of issues, and recommendations for improvement [28].

CHAPTER III

RELATED WORK

In this section, we will discuss work related to this thesis. In particular, we will look at security in Django-based applications and Kubernetes-based environments. Also, we will look at data storage security in cloud computing based on hybrid encryption schemes and zero-knowledge encryption.

Django-based applications

Aborujilah et al. [19] addressed security challenges in modern web applications, focusing on common vulnerabilities such as SQL injections, cross-site scripting, and broken authentication. It reviews, compares, and analyzes the built-in security features of various web development frameworks (e.g., Laravel, Spring Boot, Django) to protect against these vulnerabilities. The study aimed to help developers and organizations choose the most effective protection methods offered by web application frameworks, as manual protection mechanisms can increase the likelihood of attacks.

Giannopoulos et al. [1] discussed web frameworks that provide default security checks but can be vulnerable to attacks like Cross-site Scripting (XSS) and Cross-Site Request Forgery (CSRF) if developers disable these checks. Framework-specific elements make identifying such issues difficult. The authors introduced Pythia, a tool designed to analyze Django-based applications, considering all framework-specific elements and employing data-flow analysis and taint tracking techniques. Pythia is the first mechanism of its kind. The evaluation of Pythia on five open-source applications yielded positive results, identifying dangerous paths leading to vulnerabilities in four cases, often involving Django-specific features.

Kubernetes-Based Environments

Shamim et al. [15] focused on Kubernetes, an open-source software for automating computerized services management, which is susceptible to security vulnerabilities. To help practitioners secure their Kubernetes installations, the authors analyze 104 Internet artifacts and systematize knowledge related to Kubernetes security practices. They identify 11 security practices, including implementing role-based access control (RBAC) for least privilege, applying security patches to keep Kubernetes updated, and enforcing pod and network-specific security policies.

Mytilinakis [20] investigated various attacks on Kubernetes, including those targeting the engine and its components, network layer attacks like MITM and DNS spoofing, attacks on containers within pods, and Infrastructure as Code vulnerabilities. The author suggested defense measures such as using the kube-bench tool, implementing network policies and service meshes, employing the Clair scanner for container security, and applying Pod Security Policies to prevent the deployment of vulnerable code.

Hybrid Encryption Schemes

Sarkar and Kumar [8] discussed cloud computing, which allows organizations to adopt information technology without upfront investment, offering on-demand computational infrastructure and storage services. However, data security is a significant challenge in cloud computing due to the possibility of unauthorized access through virtual machines. The authors proposed a new framework based on Hybrid Encryption Schemes to address this issue, which can efficiently encrypt and retrieve data. Performance evaluation and validation demonstrate that the proposed architecture is feasible, scalable, and efficient for ensuring data security in cloud computing.

Zero-Knowledge Encryption

Luchs et al. [13] discussed the benefits of cloud computing for businesses, such as cost savings and scalable computing, but also highlighted the risk of data leakage due to challenges in verifying the confidentiality and integrity of cloud services. Zero-knowledge data encryption is presented as a solution to address these security concerns. A zero-knowledge system ensures that the system does not know user data content, using a private key known only to the user for encryption before storing on the server. This approach, combined with other security measures, restricts data decryption to the original user and enhances security. The paper analyzed the comparative advantages of zero-knowledge encryption and traditional security schemes in cloud data storage.

Overcoming the Gap in Related Work

Our related work covers various security aspects in web applications like Django, Kubernetes clusters, and cloud-based storage systems. However, there are some gaps and potential areas for improvement. For example, the literature discusses Django [19,1] and Kubernetes [15, 20] security separately. However, we observe a lack of research on the specific combination of Kubernetes, Django, and zero-knowledge encryption in a single application. Our work bridges this gap by showcasing how Django's built-in security features can be effectively combined with Kubernetes deployment, providing a secure web application environment. Also, the related work mainly provided a theoretical analysis of zero-knowledge encryption in cloud storage without discussing practical implementation details. On the other hand, our work demonstrates a real-world implementation of zero-knowledge encryption combined with other security measures in a Django-based application running on Kubernetes clusters.

CHAPTER IV

PROBLEMS AND METHODS

We employed modern technologies and security measures to ensure data security and privacy in the cloud. In addition, we used the Django platform. This free, open-source, Python-based web framework follows the model-template-views (MTV) architectural pattern for building the website and facilitating rapid development.

We created a database using Python and PostgreSQL within a Virtual Private Cloud (VPC) to enhance the site's security. In addition, robust security measures were employed to protect the data, making unauthorized changes difficult. Finally, the connection between the cloud and application databases was established as specified in the (.env.prod file) (Figure 2).

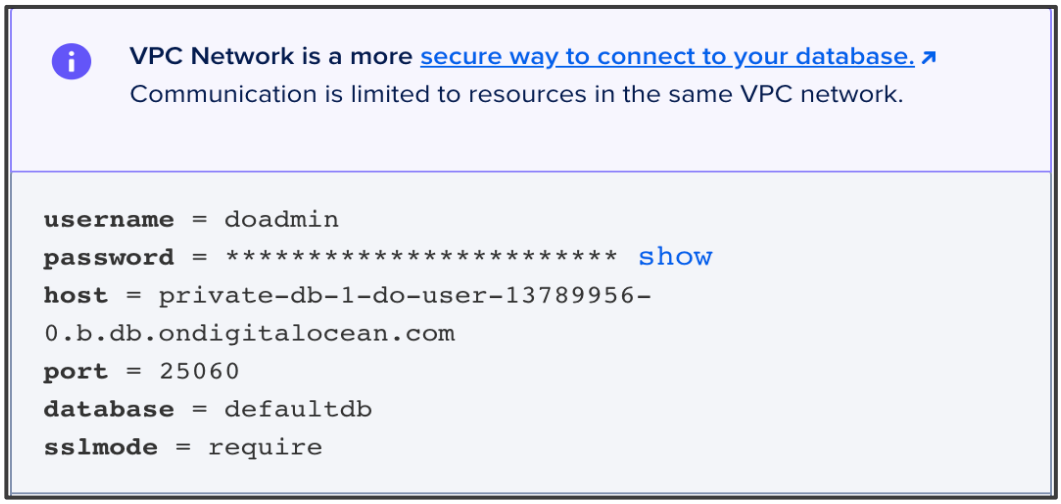


Figure 2. The database security information with VPC.

To facilitate secure and confidential file uploads, we connected the cloud storage in the application with Content Delivery Networks (CDN). This CDN file allowed for the efficient transfer of fixed files, such as storage, to the cloud and optimize storage usage. The CDN file has following: AWS_S3 settings:

AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_STORAGE_BUCKET_NAME, AWS_S3_ENDPOINT_URL, AWS_S3_OBJECT_PARAMETERS, DEFAULT_FILE_STORAGE, and STATICFILES_STORAGE.

We utilized Docker to package the project into small, easily transportable containers. Docker is a set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers. The Docker command line utilities were used to build container images based on Dockerfiles, push/pull these images to/from remote web repositories and run/deploy containers on the host machine. (Figure 3).

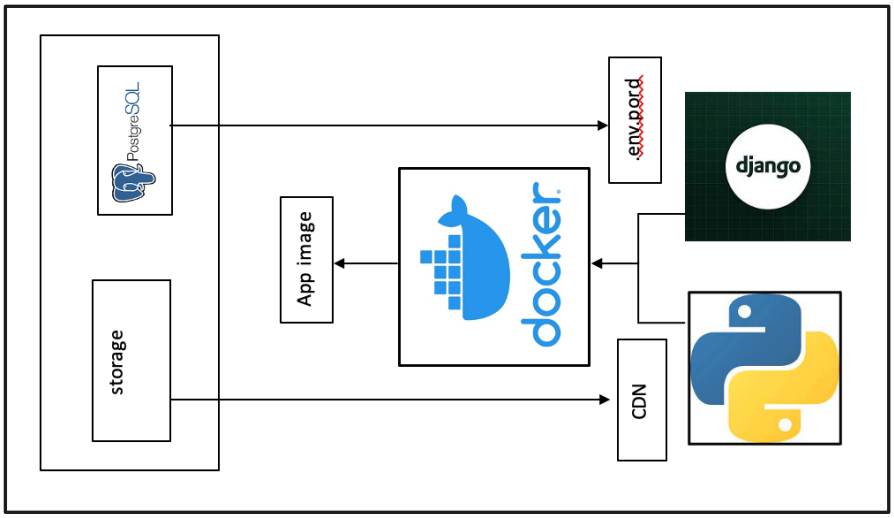


Figure 3. The diagram shows using Docker as a container for the application images.

We used Kubernetes, an open-source container orchestration system for automating software deployment, scaling, and management (Figure 4).

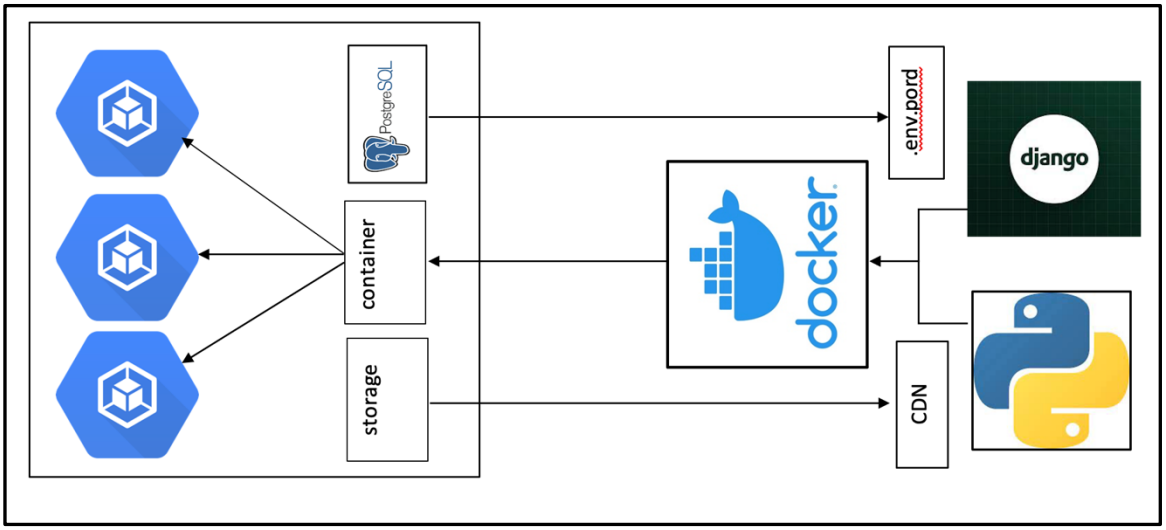


Figure 4. The full diagram shows when the application connects with nodes.

Kubernetes is what we use to distribute applications, and it provides a safe environment for publishing applications. Although in this Kubernetes environment there are many ways for configuring Kubernetes, we use deployment yaml files in the configuration (Figures 5, 6, 7, and 8):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: thesis-deployment
  labels:
    app: thesis-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: thesis-deployment
  template:
    metadata:
      labels:
        app: thesis-deployment
    spec:
      serviceAccountName: django-k8s-web-serviceaccount
      containers:
      - name: my-thesis-upload
        image: registry.digitalocean.com/container-1/my-thesis-upload:latest
        imagePullPolicy: Always
        envFrom:
          - secretRef:
              name: thesis-secure
        env:
          - name: PORT
            value: "8004"
        ports:
        - containerPort: 8004
      imagePullSecrets:
        - name: container-1
```

Figure 5. Configuration deployment yaml file.

1. Deployment (thesis-deployment) (Figure 5):
    - This resource creates a "thesis-deployment" deployment with three replicas for high availability.
    - The deployment uses the "django-k8s-web-serviceaccount" service account for authentication and authorization.

- The application container, named "my-thesis-upload," is based on the image "registry.digitalocean.com/container-1/my-thesis-upload:latest."
- The image is pulled with the "Always" pull policy, ensuring the latest version is used.
- The container exposes port 8004 and retrieves environment variables from the "thesis-secure" secret.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: thesis-service
spec:
  type: ClusterIP
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8004
    - name: https
      protocol: TCP
      port: 443
      targetPort: 8004

  selector:
    app: thesis-deployment
```

Figure 6. Configuration service yaml file.

2. Service (thesis-service) (Figure 6):

- This resource creates a ClusterIP service named "thesis-service" to expose the deployment internally within the Kubernetes cluster.
- The service maps port 80 to the target port 8004 for HTTP and port 443 to the target port 8004 for HTTPS.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: django-k8s-web-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/content-security-policy: "default-src 'self'; script-src
    'self' https://cdn.jsdelivr.net https://nyc3.digitaloceanspaces.com
    https://trusted.script.source; img-src 'self' data:
    https://nyc3.digitaloceanspaces.com https://trusted.image.source
    style-src 'self' https://cdn.jsdelivr.net https://nyc3.digitaloceanspaces.com
    https://trusted.style.source; font-src 'self' https://cdn.jsdelivr.net
    https://nyc3.digitaloceanspaces.com; object-src 'none'; frame-ancestors
    'none'; connect-src 'self';"
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  tls:
  - hosts:
    - thesis.hospitaltest.site
    - www.thesis.hospitaltest.site
    secretName: django-k8s-web-tls
  rules:
  - host: thesis.hospitaltest.site
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: thesis-service
            port:
              name: https
  - host: www.thesis.hospitaltest.site
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: thesis-service
            port:
              name: https
```

Figure 7. Configuration ingress yaml file.

3. Ingress (django-k8s-web-ingress) (Figure 7):
    - This resource creates an ingress controller to expose the application externally.
    - It uses the NGINX ingress controller and enforces a Content Security Policy (CSP) for improved security.
    - It forces SSL redirection, meaning all requests will be redirected to HTTPS.
    - The ingress is configured to route traffic for "thesis.hospitaltest.site" and "www.thesis.hospitaltest.site" to the "thesis-service" on the HTTPS port.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: django-k8s-web-rolebinding
subjects:
- kind: ServiceAccount
  name: django-k8s-web-serviceaccount
  namespace: default
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: django-k8s-web-serviceaccount
  namespace: default
```

Figure 8. Configuration Role, RoleBinding, and ServiceAccount yaml files.

4. Role (pod-reader) (Figure 8):
    - This resource defines a role named "pod-reader" that allows reading ("get", "watch", "list") access to the "pods" resource in the cluster.
5. RoleBinding (django-k8s-web-rolebinding) (Figure 8):
    - This resource binds the "pod-reader" role to the "django-k8s-web-serviceaccount" service account, granting the specified permissions to the service account.
6. ServiceAccount (django-k8s-web-serviceaccount) (Figure 8):
    - This resource creates a service account named "django-k8s-web-serviceaccount" in the "default" namespace for the application to use for authentication and authorization.

After configuring the YAML files, the deployment functions effectively, allowing the application to be launched seamlessly in the real world. In the event of human errors, such as accidentally deleting nodes, alternative nodes are automatically generated due to the command in the file, which specifies the number of nodes needed for the application deployment (Figure 9). In addition, each node contains all application images, ensuring settings remain unchanged.

thesis-deployment-9bd9fb459-4mblp	1/1	Running
thesis-deployment-9bd9fb459-fgccw	1/1	Running
thesis-deployment-9bd9fb459-fxfhq	1/1	Running

Figure 9. Nodes of the application.

With the improved deployment process and security in the cloud, it became possible to use the cloud to allow users to upload files easily (Figure 10).

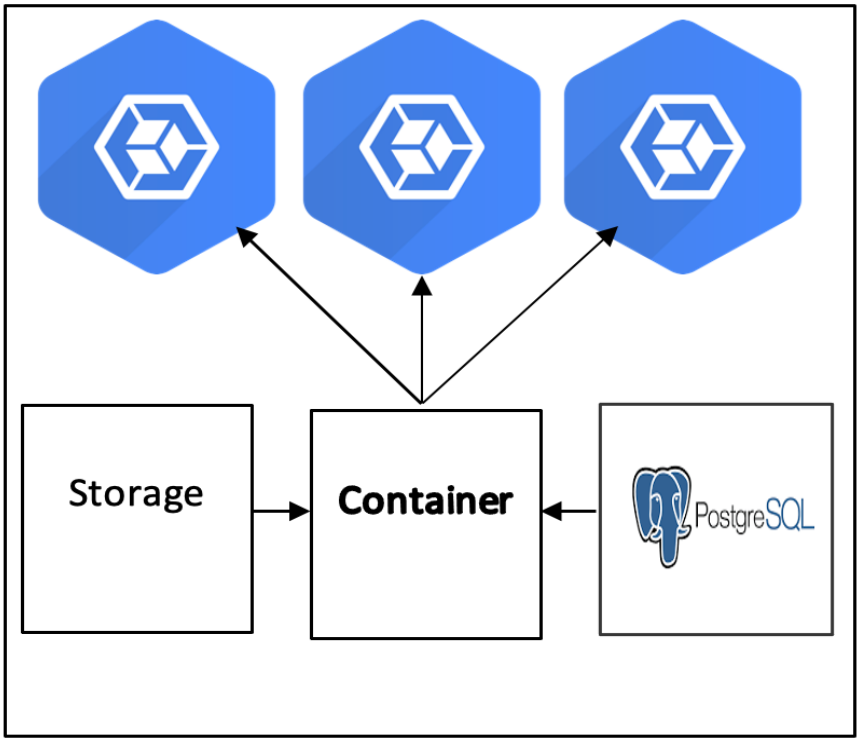


Figure 10. This figure shows the connection between database storage and the nodes.

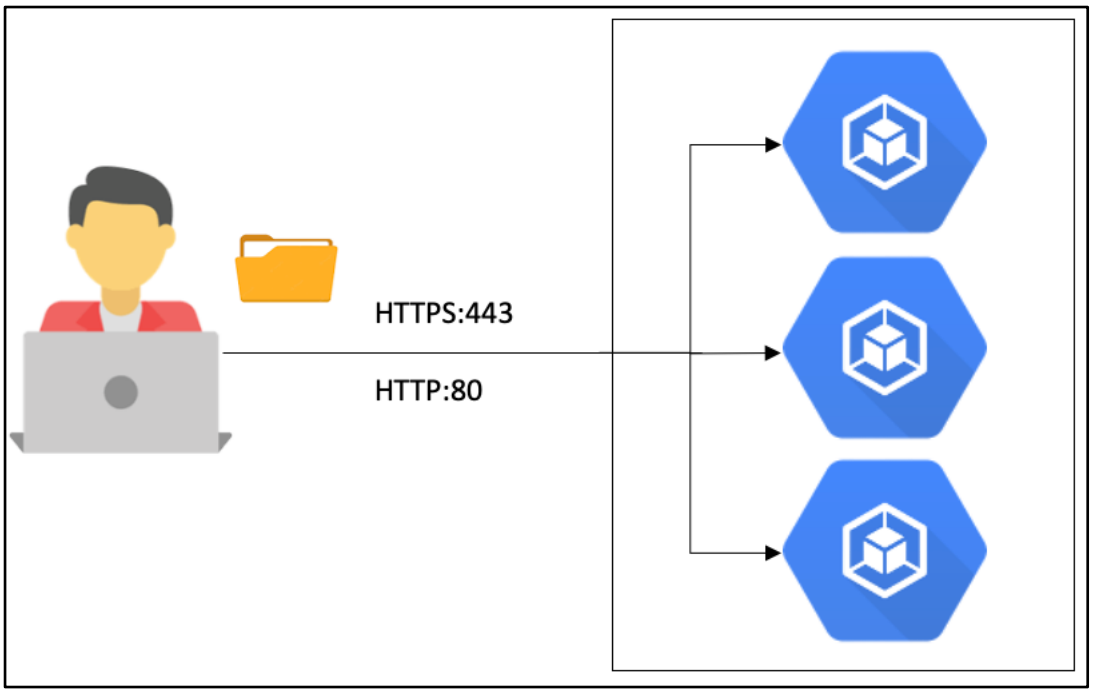


Figure 11. This figure shows that the user can connect with any node using Helm https.

Comprehensive and detail-oriented security measures have been implemented to protect all communication pathways between the application, nodes, database, and storage systems (Figure 11). These security measures have been achieved through the deployment of firewalls, which serve to permit only specified IP addresses or ports to establish connections with other components within the network. Such a well-planned security strategy effectively minimizes the risk of unauthorized access, safeguarding the integrity of the entire system and its sensitive data.

Indeed, the simplicity with which user files can be uploaded poses a significant challenge in establishing distinct user roles to deter unauthorized access between users. Furthermore, implementing robust authentication and authorization mechanisms is essential within the Django framework to guarantee the security and privacy of user data and interactions. Incorporating Role-Based Access Control (RBAC) within the website is

a vital security measure to protect sensitive information and safeguard the associated components. By employing RBAC, access to system resources and data is limited to authorized individuals according to their designated roles, substantially mitigating the risk of unauthorized access or data breaches. This security strategy bolsters the overall protection and privacy of the website's infrastructure and the data it oversees, creating a secure environment for users and stakeholders.

Where users wish to upload a file, they may opt to utilize any available node, as all nodes maintain seamless connectivity with both the storage system and the database, as illustrated in (Figures 11, 12, and 15). If a user selects the option to upload a file without implementing encryption, the file will be stored accordingly. The database will register the user's choice of uploading in an unencrypted manner, as depicted in (Figure 15). This flexible design allows for diverse user requirements while comprehensively understanding user activities and preferences.

Hybrid encryption is used, considered more secure, and is done in two ways. The first one comes from the cloud or server side, which allows the key to be used in the cloud without problems and stored in a specific way in the database (Figure 16). Decoding is easy using this approach. Another approach is to apply encryption user-side with ZKE, which is known as zero-knowledge encryption (Figures 17, 18, 19, and 20).

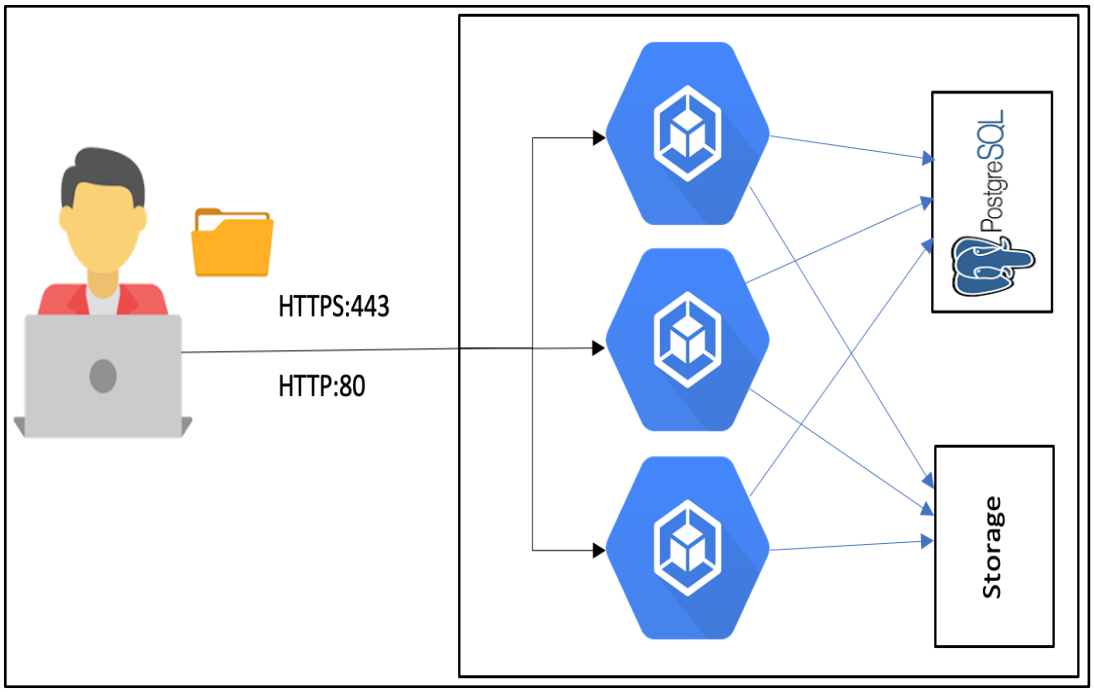


Figure 12. This diagram shows the first step of uploading a file to the server by the user.

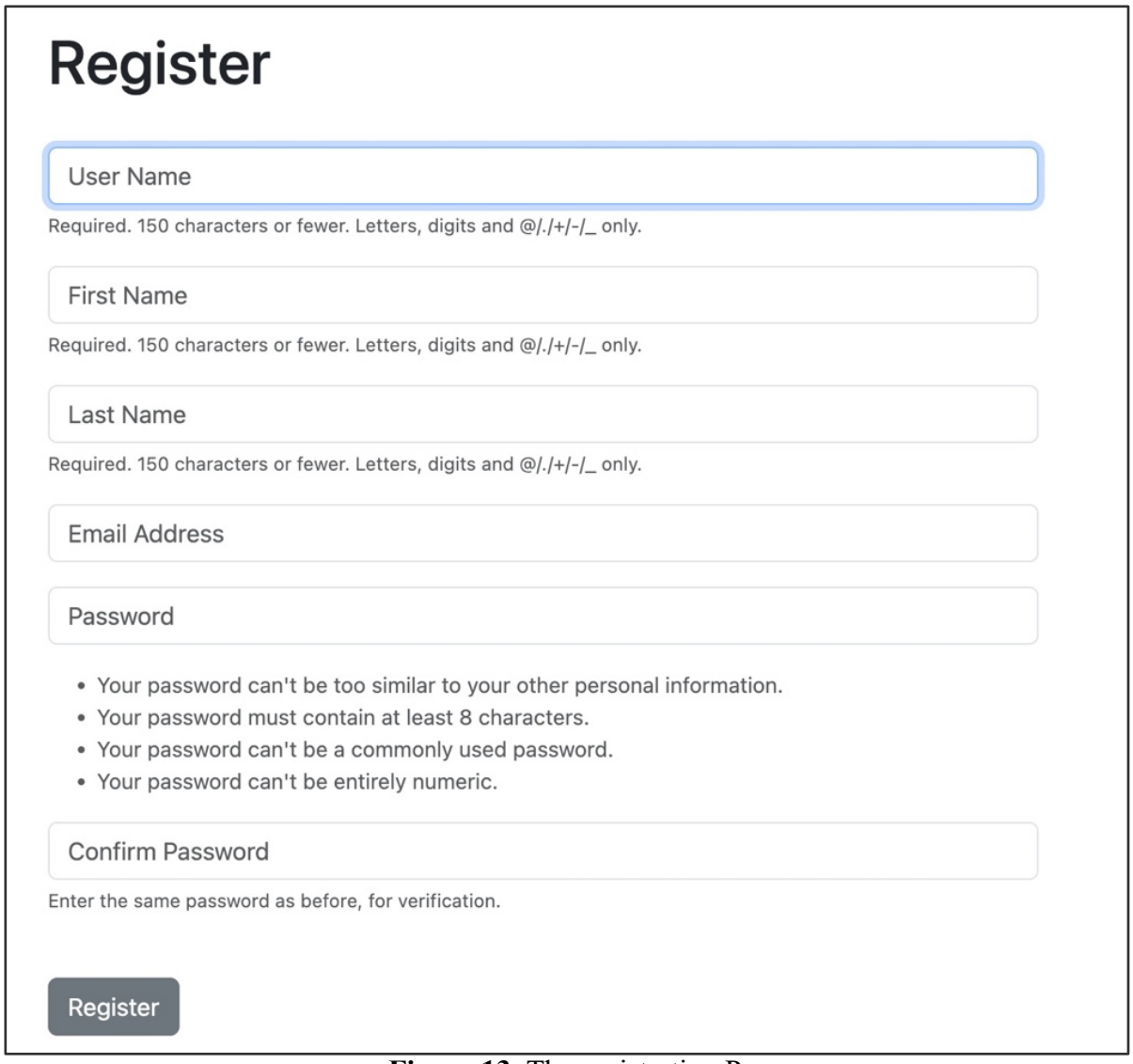


Figure 13. The registration Page.

To utilize the application, users must complete a registration process that entails submitting specific personal information, including their name, username, password, and unique email address (Figure 13). Following successful registration, users may log in anytime to upload additional files. When users provide input inconsistent with their previously registered information, the system will notify them appropriately (Figure 14).

Figure 14. The login Page.

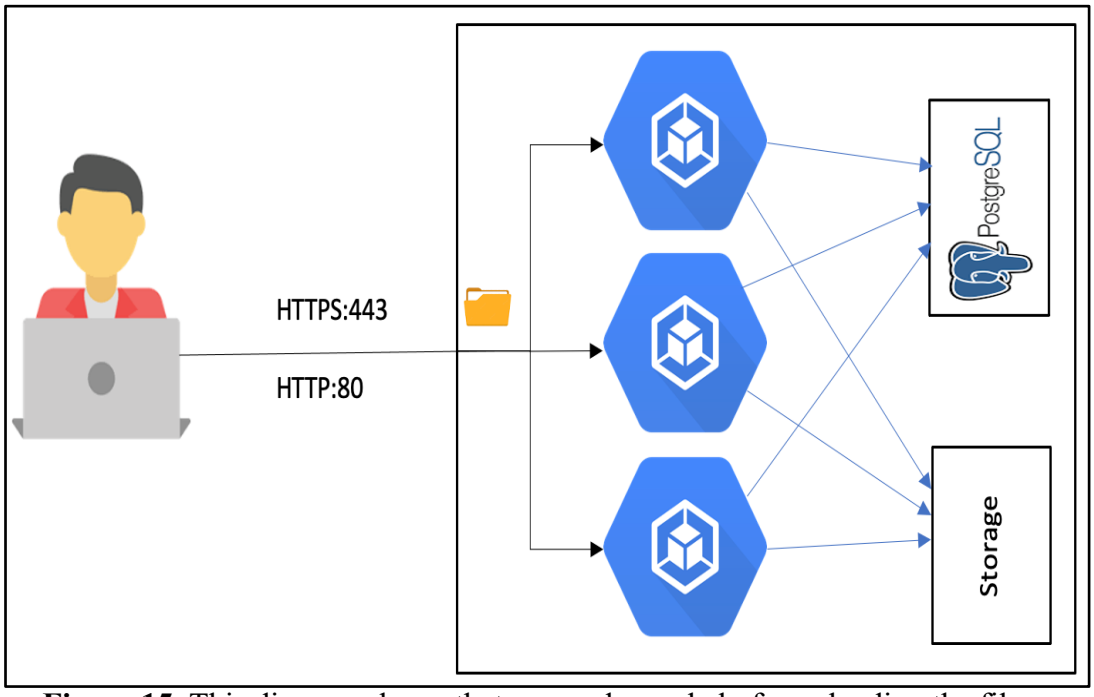


Figure 15. This diagram shows that any node can help for uploading the file.

Figure 16. The three options for encryption of a file when uploading.

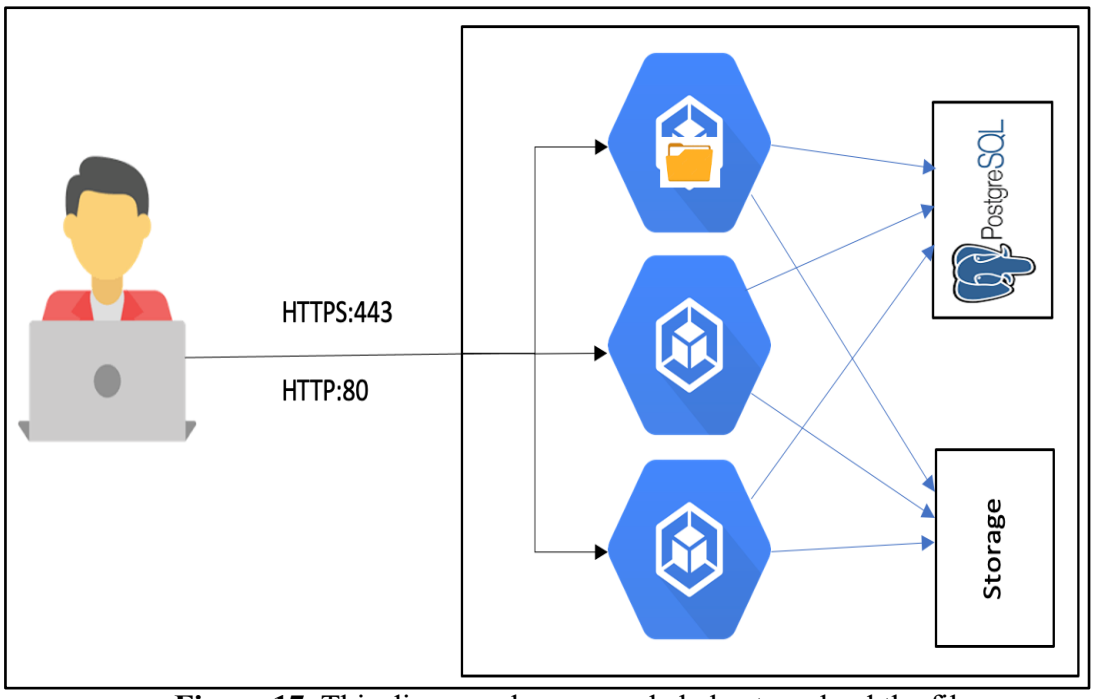


Figure 17. This diagram shows a node helps to upload the file.

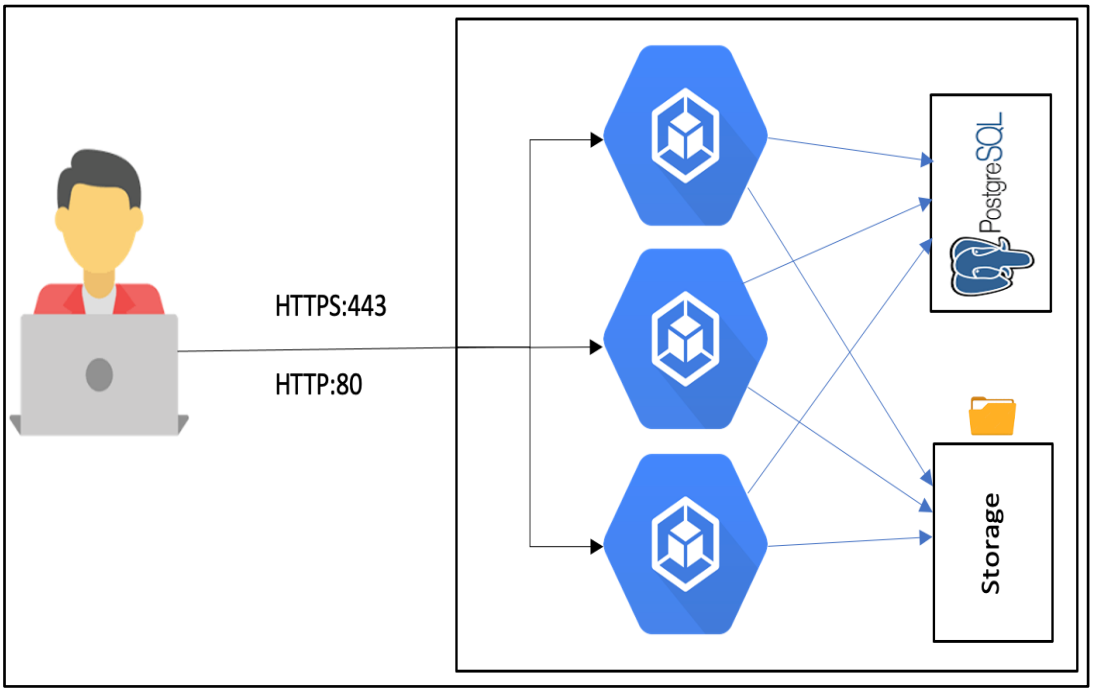


Figure 18. This diagram shows the file is stored without encryption.

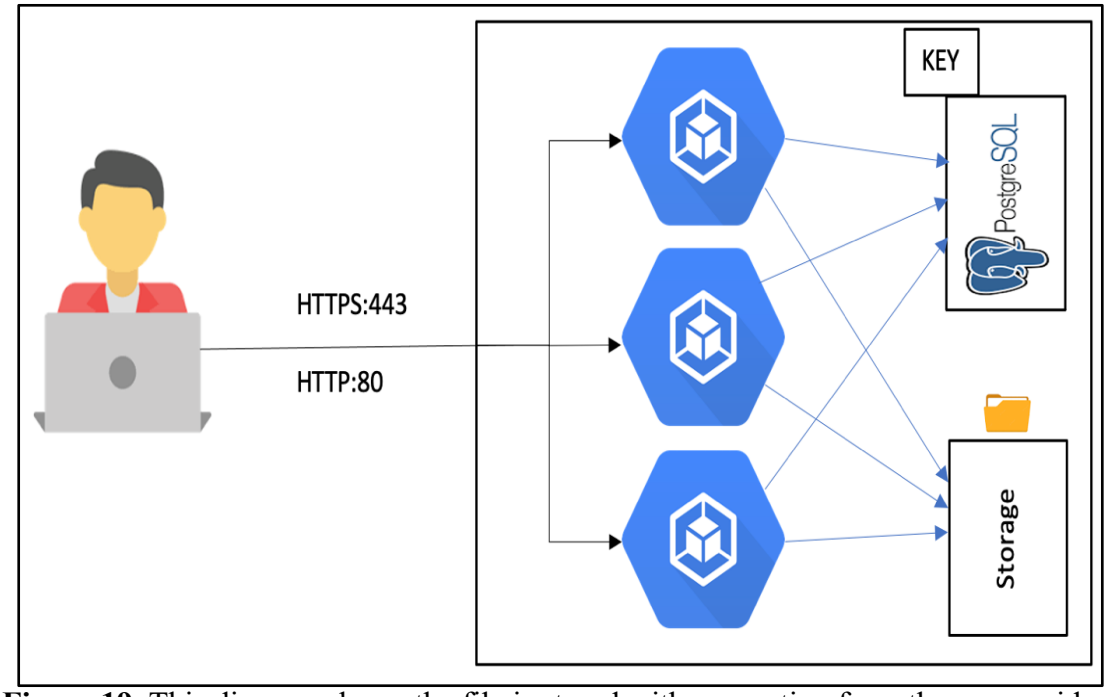


Figure 19. This diagram shows the file is stored with encryption from the server-side.

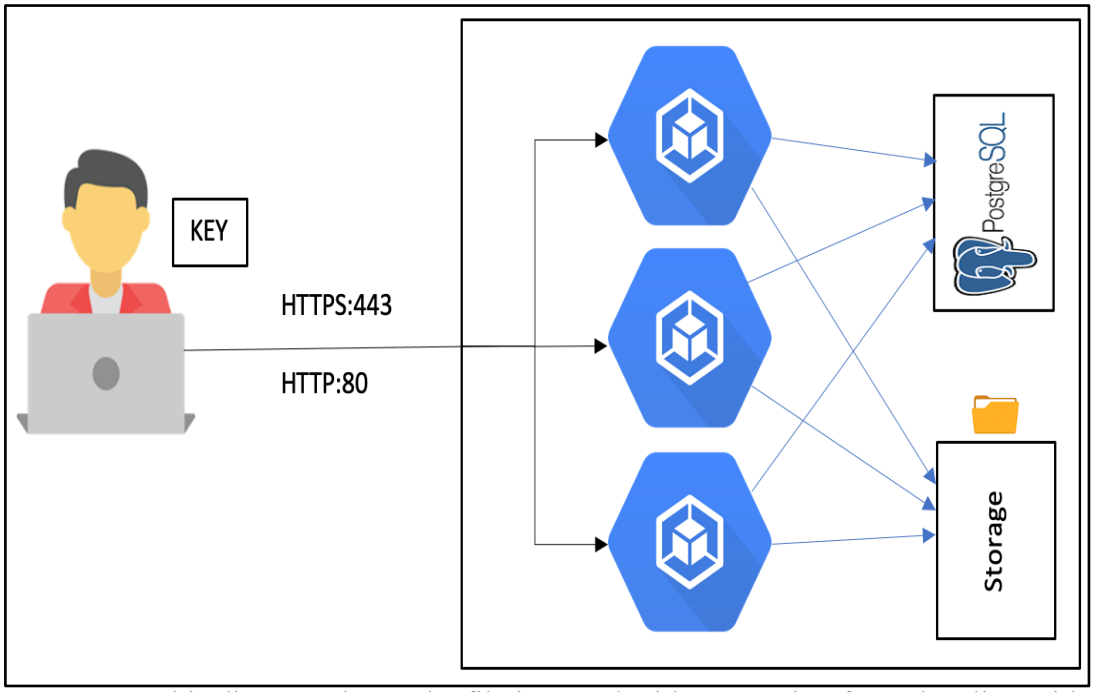


Figure 20. This diagram shows the file is stored with encryption from the client-side.

CHAPTER V

RESULTS

Depending on the method, we have the results of testing the system differently. After testing the encryption and decryption using both sides, we can say all of them protect the data and are not accessible for attackers to use the key for the files encrypted by the server side (Table 1).

Table 1. An example of how to insert a file into the database.

Name	Key	Key	Is-encryption	Option type
images/alipic.jpg	NULL	NULL	0	no-encryption
images/FinalPaper.pdf	NULL	NULL	0	no-encryption
images/alipic.jpg	NULL	NULL	1	client-side
images/FinalPaper. pdf	NULL	NULL	1	client-side
images/alipi.jpg	x'4c95b445d2e25f7e..	x'2984a77824055af650dle.	1	server-side
images/FinalPaper.pdf	x'66f78cbc216eb9fba.	x'2a5b3a27aca65c9866b6b5.	1	server-side

We can see this in the user upload in the database, which shows test upload files with three levels. First we examine the encryption client-side, which means using Zero Knowledge Encryption (ZKE) (Figure 22). Using ZKE will save the data in storage and let the database this user uploads data and no key in PDF and JPG files. The second shows uploading files without encryption and saving them in storage and database without a key in PDF and JPG files. The last one is server-side encryption which offers the key to the database and the file in the repository for both files (Table 1).

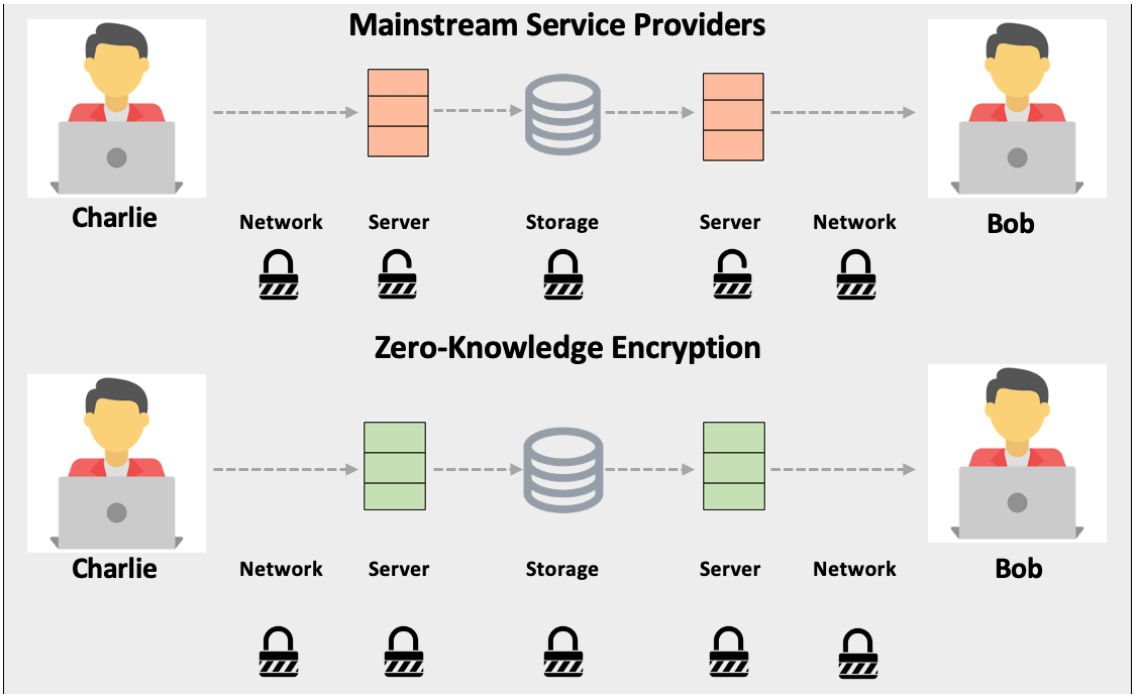


Figure 21. Zero Knowledge Encryption.

Figure 22. The results show how the file encrypted with the three options of encryptions on the List Upload Interface.

Table 2. Results of encryption methods for stored files

Name	symmetric_key	private_key	tag	iv	salt	Is-encryption	METHOD
images/ali.jpg	NULL	NULL	NULL	NULL	NULL	0	no-encryption
images/ali.jpg	NULL	NULL	NULL	NULL	NULL	1	client-side
images/ali.jpg	x'4c2e25f7e..	x'782405dle.	x'29840dle.	x'2984adle.	NULL	1	server-side

As users interact with the website to upload files, they are presented with three distinct options for the uploading process, as demonstrated in (Figure 13). The responsibility of selecting whether to upload files without encryption or utilizing encryption mechanisms rests solely with the user. This approach offers individuals the autonomy to make informed decisions based on their preferences and security requirements, ensuring that the website caters to diverse user needs while maintaining high adaptability.

In Table 2, the first row represents a file encrypted using the client-side encryption method. In this case, the symmetric_key, private_key, tag, and iv fields are set to NULL since these values are not stored on the server in a zero-knowledge encryption scheme. The Is-encryption field is set to 1 (True), and the ENCRYPTION_METHOD field is set to 'client-side'. The second row represents a file that has not been encrypted. In this case, all the encryption-related fields (symmetric_key, private_key, tag, and iv) are set to NULL, as they are unnecessary for an unencrypted file. The Is-encryption field is set to 0 (False), and the ENCRYPTION_METHOD field is set to 'no-encryption'. The third row represents a file encrypted using the server-side encryption method. In this case, the symmetric_key, private_key, tag, and iv fields have non-null values used in the encryption and decryption processes. The Is-encryption field is set to 1 (True), and the ENCRYPTION_METHOD

field is set to 'server-side'. As shown in (Figure 23), the user can use any of these three options to upload the file, and the user can choose which option is used.

We have leveraged Content Delivery Networks (CDNs) to secure stable files in the cloud, optimizing the application of Kubernetes and eliminating the need for Persistent Volumes (PVs) and Persistent Volume Claims (PVCs). File stability has been achieved by integrating a CDN with our website hosted on Kubernetes. Our Kubernetes cluster distributes the website's content via HTTPS and incorporates the CDN provider's servers as an additional caching layer.

Monitoring

This section presents a comparative analysis of two distinct Kubernetes YAML file configurations and their impact on security. Website A employs a high-security setup, whereas Website B utilizes the default settings. Three monitoring mechanisms are deployed to examine the differential security outcomes for each website.

A comprehensive security assessment was conducted on Website A, revealing an 8% failure rate in terms of security performance. The discrepancy between Website A and Website B, demonstrating a 7% failure rate, highlights the importance of constant evaluation and refinement of security measures in website development (Figures 24 and 25).

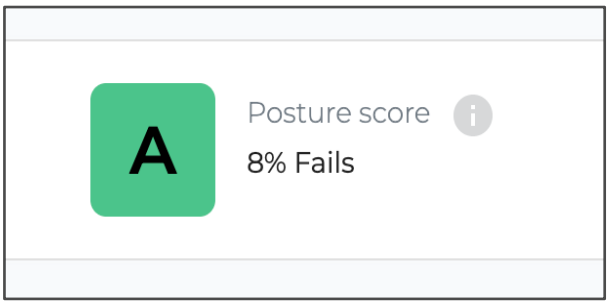


Figure 23. Monitoring from Datree website A.

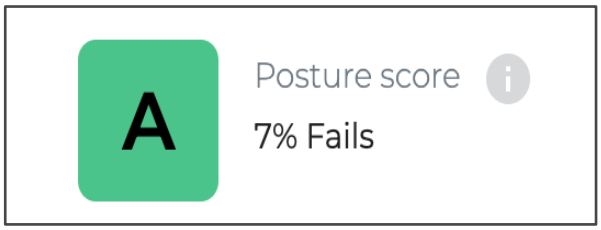
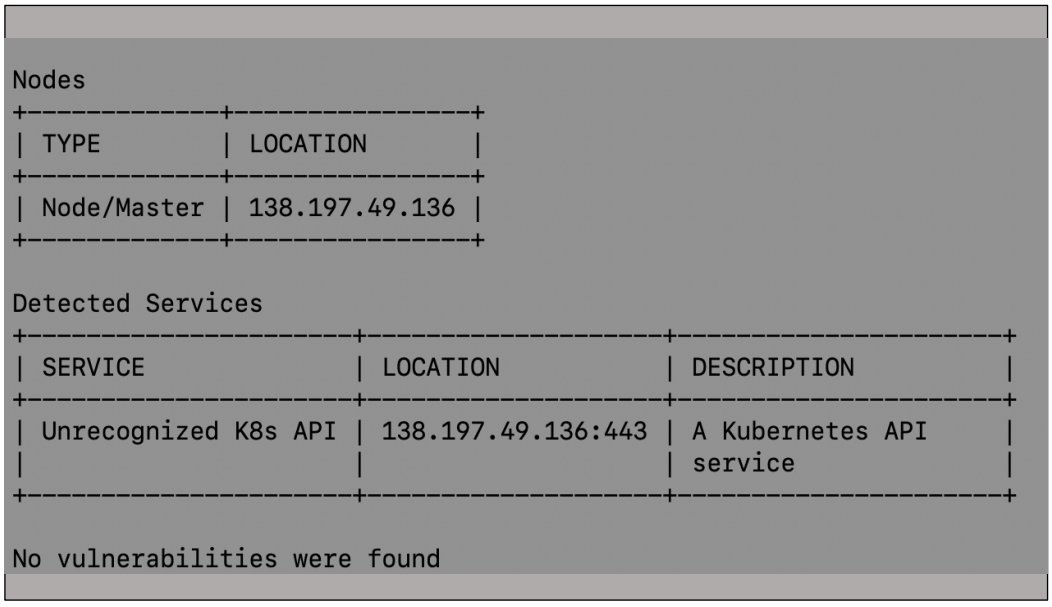


Figure 24. Monitoring from Datree website B.

The Kube-hunter tool was utilized to investigate possible security vulnerabilities associated with the websites. The analysis of Website A revealed a complete lack of exploitable vulnerabilities. In contrast, the comparative study of another website detected vulnerabilities using Kube-hunter, emphasizing the significance of a well-established security framework (Figures 26 and 27).

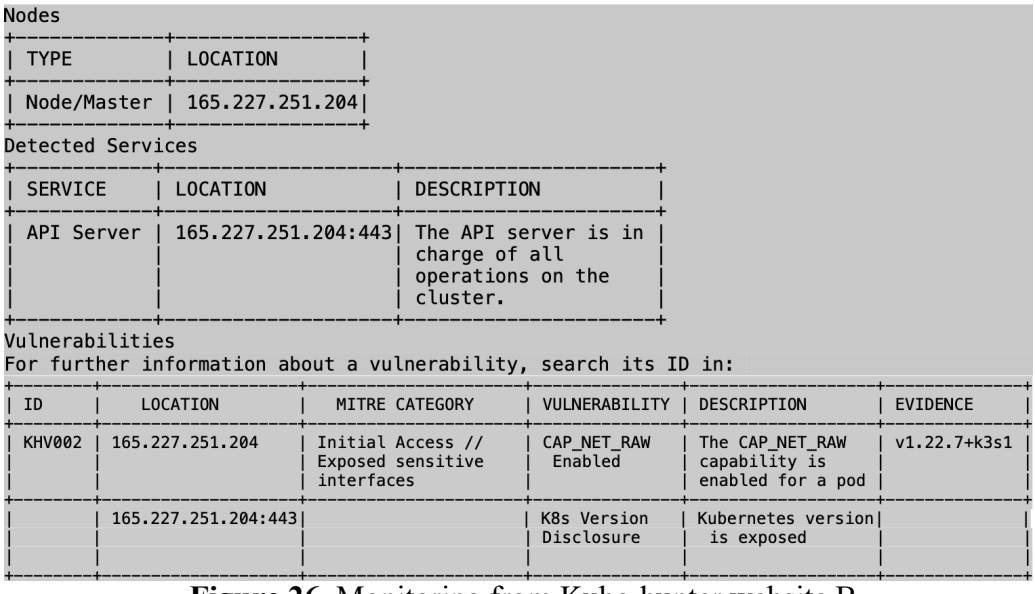


```
Nodes
+-------------+----------------+
| TYPE        | LOCATION       |
+-------------+----------------+
| Node/Master | 138.197.49.136 |
+-------------+----------------+

Detected Services
+----------------------+---------------------+----------------------+
| SERVICE              | LOCATION            | DESCRIPTION          |
+----------------------+---------------------+----------------------+
| Unrecognized K8s API | 138.197.49.136:443 | A Kubernetes API     |
|                      |                     | service              |
+----------------------+---------------------+----------------------+

No vulnerabilities were found
```

Figure 25. Monitoring from Kube-hunter website A.



```
Nodes
+-------------+----------------+
| TYPE        | LOCATION       |
+-------------+----------------+
| Node/Master | 165.227.251.204|
+-------------+----------------+
Detected Services
+------------+--------------------+-----------------------+
| SERVICE    | LOCATION           | DESCRIPTION           |
+------------+--------------------+-----------------------+
| API Server | 165.227.251.204:443| The API server is in  |
|            |                    | charge of all         |
|            |                    | operations on the     |
|            |                    | cluster.              |
+------------+--------------------+-----------------------+
Vulnerabilities
For further information about a vulnerability, search its ID in:
+-------+-----------------+--------------------+---------------+-------------------+-------------+
| ID    |    LOCATION     |   MITRE CATEGORY   | VULNERABILITY | DESCRIPTION       | EVIDENCE    |
+-------+-----------------+--------------------+---------------+-------------------+-------------+
| KHV002 | 165.227.251.204 | Initial Access //  | CAP_NET_RAW   | The CAP_NET_RAW   | v1.22.7+k3s1 |
|       |                 | Exposed sensitive  |  Enabled      | capability is     |             |
|       |                 | interfaces         |               | enabled for a pod |             |
+-------+-----------------+--------------------+---------------+-------------------+-------------+
|       | 165.227.251.204:443|                 | K8s Version   | Kubernetes version|             |
|       |                 |                    | Disclosure    |  is exposed       |             |
|       |                 |                    |               |                   |             |
+-------+-----------------+--------------------+---------------+-------------------+-------------+
```

Figure 26. Monitoring from Kube-hunter website B.

The Mozilla Observatory, a prominent tool for assessing security configurations, was used to analyze the project's website. This evaluation generated an A+ rating and an exceptional score of 125/100, indicative of an optimal security configuration. However, a second website assessed using the same tool garnered a C+ rating, with a score of 60/100, passing 9 out of the 11 tests. The disparity in security ratings is attributed to factors such as robust encryption, data protection protocols, and advanced security measures like Content Security Policy (CSP) and HTTP Strict Transport Security (HSTS) (Figures 28 and 29).

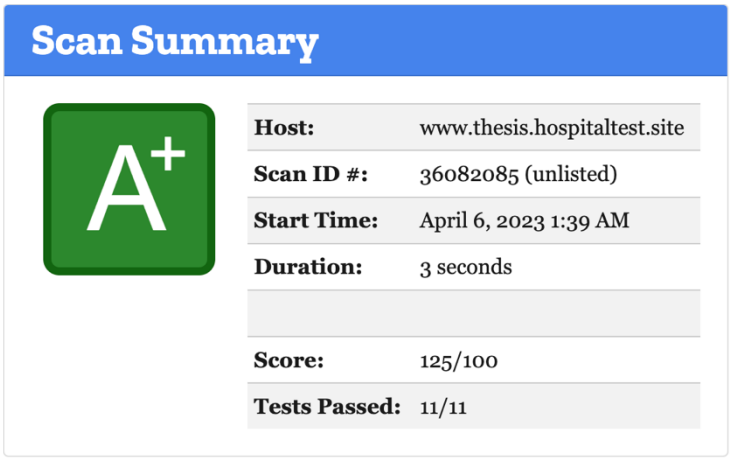


Figure 27. Monitoring from Mozilla Observatory website A.

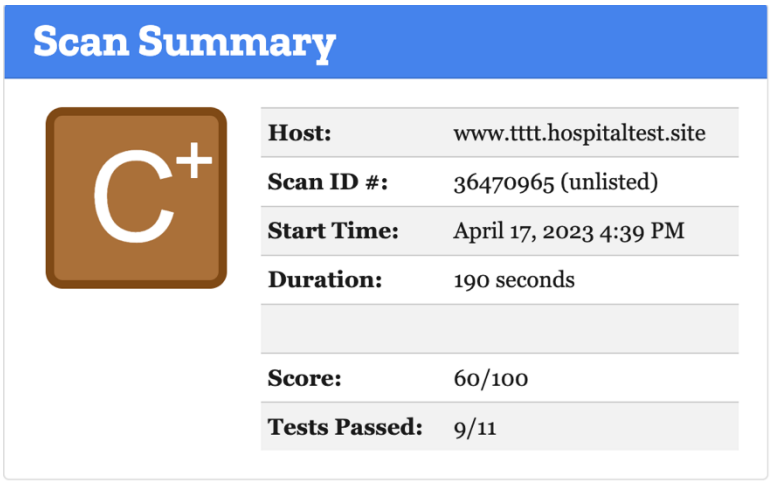


Figure 28. Monitoring from Mozilla Observatory website B.

If an attacker attempts to compromise a website, systematic efforts to gain unauthorized access to sensitive pages or functions, such as the login page, may occur. The server detects malicious activity, temporarily blocks the attacker's IP address, and denies further access to the targeted login page (Figure 30). This defense mechanism mitigates the risk of unauthorized access and helps maintain the website's integrity (Table 3).

```
[17/Apr/2023 19:15:48] "GET /login HTTP/1.1" 301 0
[17/Apr/2023 19:15:48] "GET /login/ HTTP/1.1" 200 8917
[17/Apr/2023 19:15:48] "GET /login HTTP/1.1" 301 0
[17/Apr/2023 19:15:48] "GET /login/ HTTP/1.1" 200 8917
[17/Apr/2023 19:15:49] "GET /login HTTP/1.1" 301 0
WARNING:root:IP 127.0.0.1 temporarily blocked due to suspicious activity
[17/Apr/2023 19:15:49] "GET /login/ HTTP/1.1" 200 8917
[17/Apr/2023 19:15:49] "GET /login HTTP/1.1" 301 0
Forbidden: /login/
WARNING:django.request:Forbidden: /login/
[17/Apr/2023 19:15:49] "GET /login/ HTTP/1.1" 403 34
[17/Apr/2023 19:15:50] "GET /login HTTP/1.1" 301 0
Forbidden: /login/
WARNING:django.request:Forbidden: /login/
[17/Apr/2023 19:15:50] "GET /login/ HTTP/1.1" 403 34
[17/Apr/2023 19:15:50] "GET /login HTTP/1.1" 301 0
Forbidden: /login/
WARNING:django.request:Forbidden: /login/
[17/Apr/2023 19:15:50] "GET /login/ HTTP/1.1" 403 34
[17/Apr/2023 19:15:51] "GET /login HTTP/1.1" 301 0
Forbidden: /login/
WARNING:django.request:Forbidden: /login/
[17/Apr/2023 19:15:51] "GET /login/ HTTP/1.1" 403 34
[17/Apr/2023 19:15:51] "GET /login HTTP/1.1" 301 0
Forbidden: /login/
WARNING:django.request:Forbidden: /login/
[17/Apr/2023 19:15:51] "GET /login/ HTTP/1.1" 403 34
[17/Apr/2023 19:15:52] "GET /login HTTP/1.1" 301 0
Forbidden: /login/
WARNING:django.request:Forbidden: /login/
[17/Apr/2023 19:15:52] "GET /login/ HTTP/1.1" 403 34
[17/Apr/2023 19:15:52] "GET /login HTTP/1.1" 301 0
```

Figure 29. Shows the attack and the message with the IP.

Table 3. IP address blocking due to excessive requests

NO.	ip_address	block_reason	block_time
1	127.0.0.1	Exceeded requests per minute threshold.	2023-04-17 19:22

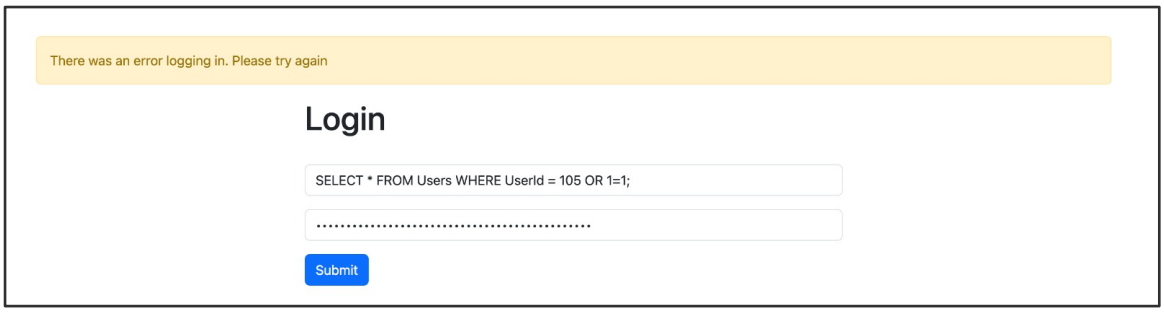


Figure 30. The login page with a warning message.

The authentication mechanism ensures secure management of user accounts and reinforces the application's overall security by safeguarding against unauthorized access. For example, when attempting to use an SQL query to log in, the system prevents it and displays, "There was an error logging in. Please try again." (Figure 31).

Benefits of using Django

Django is a high-level Python web framework that enables the rapid development of secure and maintainable websites. Using Django for your website pages offers several benefits:

1. Faster Development: Django follows the "batteries included" philosophy, which includes many built-in features that simplify and speed up the development process, such as a powerful ORM (Object-Relational Mapping), form handling, authentication, and more.
2. Scalability: Django is designed to handle various applications, from small-scale personal projects to large-scale enterprise applications. Its modular and reusable components make it easier to scale your website as your user base or feature set grows.

3. Security: Django strongly focuses on security, offering built-in protection against common web vulnerabilities like cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection. The framework also provides robust user authentication and authorization features.
4. Maintainability: Django promotes clean, modular code use by adhering to the DRY (Don't Repeat Yourself) and lose coupling principles. Using DRY encourages developers to write reusable and maintainable code, making it easier to update and extend your website over time.
5. Wide Ecosystem: Django's large and active community contributes to a broad ecosystem of reusable third-party applications, packages, and plugins means you can often find pre-built solutions for everyday tasks or requirements, further speeding up development.
6. Clear Documentation: Django is known for its extensive and well-organized documentation, which makes it easier for developers to learn and use the framework effectively.
7. Admin Interface: Django includes a built-in admin interface, which provides a convenient way to manage your application's data and perform administrative tasks without additional development.
8. Customizability: While Django comes with many out-of-the-box features, it also allows for extensive customization, ensuring you can build a website that meets your needs.

9. Portability: Django is platform-agnostic, meaning it can run on any operating system that supports Python, making it easy to migrate your website to different environments as needed.

Benefits of using Kubernetes

1. Scalability: Kubernetes simplifies the process of scaling applications up or down based on demand, allowing for efficient use of resources and handling varying workloads.
2. High availability: Kubernetes automatically distributes containers across multiple nodes, ensuring that applications remain available even if a node fails. This promotes a robust and fault-tolerant infrastructure.
3. Simplified deployment and management: Kubernetes streamlines the deployment process by managing the lifecycle of containerized applications, enabling rolling updates, automatic rollbacks, and version control.
4. Service discovery and load balancing: Kubernetes provides built-in service discovery and load balancing, which improves application performance and resilience by distributing traffic across multiple instances.
5. Resource efficiency: Kubernetes optimizes resource utilization by intelligently scheduling and allocating resources based on application requirements, resulting in cost savings.
6. Self-healing: Kubernetes can detect and replace failed containers, reschedule containers on failed nodes, and automatically scale applications based on resource constraints, promoting system stability.

7. Extensibility: Kubernetes can be extended with custom resource definitions, plugins, and a rich ecosystem of third-party tools, enabling customization and integration with existing infrastructure.
8. Container-centric approach: Kubernetes uses a container-based approach, which allows for consistent deployment and management of applications across various environments, promoting developer productivity and operational efficiency.
9. Strong community and ecosystem: Kubernetes is supported by a vibrant community and ecosystem of tools, libraries, and services, making it a popular choice for container orchestration.
10. Multi-cloud and hybrid-cloud support: Kubernetes enables you to run applications on any cloud platform or even on-premises, providing flexibility and reducing vendor lock-in.

The tabular analysis (Table 4) presents a comparative evaluation, delineating the differences between utilizing Kubernetes and traditional servers. This comparative assessment focuses on critical factors such as scalability, resource utilization, deployment and management, fault tolerance and high availability, load balancing, service discovery, and portability. The results indicate a superiority in employing Kubernetes over conventional server infrastructures.

The first factor, scalability, refers to the capability of a system to adapt to varying workloads by expanding or contracting its resources. In this context, Kubernetes demonstrates an enhanced ability to manage the scaling requirements of applications in contrast to standard servers.

Resource utilization is another critical aspect of system performance, which denotes the effective and efficient allocation of resources, such as CPU, memory, and storage. Again, Kubernetes excels in this area due to its container-based architecture, which enables efficient resource sharing and optimizes overall utilization compared to traditional server infrastructures.

In terms of deployment and management, Kubernetes streamlines the process by offering a unified and automated approach for deploying, scaling, and managing containerized applications, resulting in significantly reduced manual intervention and improved productivity, unlike the cumbersome processes typically associated with standard servers.

Furthermore, Kubernetes demonstrates a higher level of fault tolerance and high availability, as it inherently supports rapid detection and recovery from failures, ensuring that applications remain operational with minimal downtime. Conversely, conventional servers often necessitate manual intervention and lengthy recovery times.

Load balancing and service discovery are essential for distributing the workload among multiple servers to optimize resource usage and minimize response time. Kubernetes provides an integrated solution for these tasks, resulting in efficient traffic management and automatic service discovery. However, standard servers require additional external tools and manual configurations to achieve similar functionality.

Lastly, portability measures the ease with which an application can be transferred from one environment to another. Kubernetes facilitates portability by standardizing the deployment process and maintaining consistency across various platforms. This advantage

starkly contrasts the platform-specific dependencies and varying configurations of traditional server environments.

Table 4. Comparison between Kubernetes and standard servers

	Kubernetes	Standard servers
Scalability	Automatic, handles traffic spikes and load balancing	Manual, time-consuming, error-prone
Resource utilization	Efficient, shares resources across containers	Often underutilized, dedicated resources per app
Deployment and management	Simplified, declarative configuration, rolling updates	Complex, manual intervention, custom scripts
Fault tolerance and high availability	Automatic restarts, high availability	Manual configuration, additional tools
Load balancing and service discovery	Built-in load balancing and service discovery	Additional tools, manual configuration
Portability	Containerization, easily portable across environments	Environment-specific, more challenging

CHAPTER VI

DISCUSSION

This thesis demonstrated the effectiveness of combining Django, Kubernetes, hybrid encryption, zero-knowledge encryption, and attack prevention measures in enhancing cloud security and privacy. We have found that integrating these technologies and security measures yields significant benefits in terms of security and privacy. Furthermore, the built-in security features of Django, along with the scalability and orchestration capabilities of Kubernetes, create a robust and dynamic infrastructure that can likely adapt to various threats and challenges.

Integrating hybrid encryption and zero-knowledge encryption enhances the application's data security. Hybrid encryption provides an additional layer of protection, ensuring that sensitive information is encrypted using a combination of symmetric and asymmetric encryption algorithms. Zero-knowledge encryption, conversely, ensures that even the service providers cannot access the encrypted data, thus providing users with a higher level of privacy.

Furthermore, our work highlighted the benefits of implementing zero-knowledge encryption in cloud-based systems, emphasizing its potential to promote data privacy and its practical implementation in real-world scenarios.

In addition, we utilized a three-tier approach to data uploading, which grants customers the autonomy to decide whether to upload sensitive or non-sensitive data and determine the storage location of the encryption key—either in the cloud or retained by the user. This layered approach further strengthens data protection and empowers customers to safeguard their information.

CHAPTER VII

FUTURE WORK

Pursuing enhanced security and privacy in web applications and cloud storage is an ongoing endeavor, necessitating continuous efforts to expand our knowledge and develop innovative methods for thwarting potential attackers. In future research, we will explore the potential integration of machine learning or deep learning techniques with cloud storage systems to further improve security and privacy, which are paramount for all cloud users.

Additionally, we will investigate incorporating more zero-knowledge proof methods, which can bolster security during login, registration, authentication, and authorization processes. This approach will empower users with greater control over their data.

Furthermore, we will consider using multiple cloud storage providers to examine the impact on security enhancements, aiming to ensure optimal data protection without compromise. By diversifying storage options, we can identify best practices and develop tailored security strategies to meet the unique requirements of various cloud storage solutions.

Finally, we will consider implementing different server-side encryption methods and compare them with the current user-side encryption for various performance measures.

BIBLIOGRAPHY

[1] L. Giannopoulos, E. Degkleri, P. Tsanakas, and D. Mitropoulos, "Pythia: Identifying Dangerous Data-flows in Django-based Applications," in Proceedings of the 12th European Workshop on Systems Security, Dresden Germany: ACM, Mar. 2019, pp. 1–6. doi: 10.1145/3301417.3312497.

[2] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou, "Security and Privacy in Cloud Computing: A Survey," in 2010 Sixth International Conference on Semantics, Knowledge and Grids, Beijing, China: IEEE, Nov. 2010, pp. 105–112. doi: 10.1109/SKG.2010.19.

[3] B. Tubre and P. Rodeghero, "Exploring the Challenges of Cloud Migrations During a Global Pandemic," in 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, Australia: IEEE, Sep. 2020, pp. 784–785. doi: 10.1109/ICSME46990.2020.00090.

[4] M. Theoharidou, N. Papanikolaou, S. Pearson, and D. Gritzalis, "Privacy Risk, Security, Accountability in the Cloud," in 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, United Kingdom: IEEE, Dec. 2013, pp. 177–184. doi: 10.1109/CloudCom.2013.31.

[5] Z. Tari, X. Yi, U. S. Premarathne, P. Bertok, and I. Khalil, "Security and Privacy in Cloud Computing: Vision, Trends, and Challenges," IEEE Cloud Comput., vol. 2, no. 2, pp. 30–38, Mar. 2015, doi: 10.1109/MCC.2015.45.

[6] Z. Tari, "Security and Privacy in Cloud Computing," IEEE Cloud Comput., vol. 1, no. 1, pp. 54–57, May 2014, doi: 10.1109/MCC.2014.20.

[7] S. K. Sowmya, P. Deepika, and J. Naren, "Layers of Cloud – IaaS, PaaS and SaaS: A Survey," in 2014 (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 4477-4480

[8] M. K. Sarkar and S. Kumar, "Ensuring Data Storage Security in Cloud Computing Based On Hybrid Encryption Schemes," In 2016 Fourth International Conference On Parallel, Distributed And Grid Computing (PDGC), Waknaghat, India: IEEE, 2016, pp. 320–325. doi: 10.1109/PDGC.2016.7913169.

[9] P. P. W. Pathirathna, V. A. I. Ayesha, and W. A. T. Imihira, "Security Testing as a Service with Docker Containerization," In 2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA) pp. 1-7. IEEE.

[10] M. Moravcik, M. Kontsek, P. Segec, and D. Cymbalak, "Kubernetes - Evolution of Virtualization," in 2022 20th International Conference on Emerging eLearning

Technologies and Applications (ICETA), Stary Smokovec, Slovakia: IEEE, Oct. 2022, pp. 454–459. doi: 10.1109/ICETA57911.2022.9974681.

[11] V. B. Mahajan and S. B. Mane, “Detection, Analysis and Countermeasures for Container Based Misconfiguration Using Docker and Kubernetes,” in 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), Kochi, India: IEEE, Jun. 2022, pp. 1–6. doi: 10.1109/IC3SIS54991.2022.9885293.

[12] S. Rahul, C. Vajrala, and B. Thangaraju, “A Novel Method of Honeypot Inclusive WAF to Protect from SQL Injection and XSS,” in 2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON), Bengaluru, India: IEEE, Nov. 2021, pp. 135–140. doi: 10.1109/CENTCON52345.2021.9688059.

[13] R. Luchs and L. Sneeringer. "Zero-Knowledge Encryption in the Cloud: A Solution for the Remote File Storage," in 2018 University of Pittsburgh, Swanson School of Engineering, Conference Session: A11, Mar. 2018, pp.8069. Available: https://sites.pitt.edu/~budny/papers/8069.pdf

[14] I. Kovacevic, M. Marovic, S. Gros, and M. Vukovic, “Predicting Vulnerabilities in Web Applications Based on Website Security Model,” in 2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia: IEEE, Sep. 2022, pp. 1–6. doi: 10.23919/SoftCOM55329.2022.9911436.

[15] Md. S. Islam Shamim, F. Ahamed Bhuiyan, and A. Rahman, “XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices,” in 2020 IEEE Secure Development (SecDev), Atlanta, GA, USA: IEEE, Sep. 2020, pp. 58–64. doi: 10.1109/SecDev45635.2020.00025.

[16] W. K. Daniel, “Challenges on privacy and reliability in cloud computing security,” in 2014 International Conference on Information Science, Electronics and Electrical Engineering, Sapporo, Japan: IEEE, Apr. 2014, pp. 1181–1187. doi: 10.1109/InfoSEEE.2014.6947857.

[17] B. Zukran and M. M. Siraj, “Performance Comparison on SQL Injection and XSS Detection using Open Source Vulnerability Scanners,” in 2021 International Conference on Data Science and Its Applications (ICoDSA), Bandung, Indonesia: IEEE, Oct. 2021, pp. 61–65. doi: 10.1109/ICoDSA53588.2021.9617484.

[18] P. Tanakas, A. Ilias, and N. Polemi, “A Novel System for Detecting and Preventing SQL Injection and Cross-Site-Script,” in 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET), Cape Town, South Africa: IEEE, Dec. 2021, pp. 1–6. doi: 10.1109/ICECET52533.2021.9698

[19] A. Aborujilah, J. Adamu, S. M. Shariff, and Z. Awang Long, “Descriptive Analysis of Built-in Security Features in Web Development Frameworks,” in 2022 16th International Conference on Ubiquitous Information Management and Communication (IMCOM), Seoul, Korea, Republic of: IEEE, Jan. 2022, pp. 1–8. doi: 10.1109/IMCOM53663.2022.9721750.

[20] M. Panagiotis "Attack Methods and Defenses on Kubernetes" in 2020. Master's thesis, Piraeus University, Piraeus, Greece. Jun. 2020, http://dx.doi.org/10.26267/unipi_dione/311

[21] S. Watts and M. Raza. “SaaS vs PaaS vs IaaS: What’s The Difference & How To Choose.” https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/. Jun. 15, 2019.

[22] N. T. “What is Data Privacy in Cloud Computing? Definition, Challenges ..” Data Privacy in Cloud Computing. https://binaryterms.com/data-privacy-in-cloud-computing.html. Dec. 12, 2020.

[23] W. Chai. “What is the CIA Triad? Definition, Explanation, Examples.” TechTarget. https://www.techtarget.com/whatis/definition/Confidentiality-integrity-and-availability-CIA

[24] D. S. “Principles of Security-Tryhackme. Learn The Principles of Information.” https://medium.com/. Sep. 08, 2021. https://medium.com/@DimigraS/principles-of-security-tryhackme-fef726cf0b74

[25] OWASP Top Ten | OWASP Foundation. https://owasp.org/www-project-top-ten/. Accessed 10 Apr. 2023.

[26] I. Shakury. “kube-hunter: Hunt For Security Weaknesses Kubernetes Clusters.” GitHub. Sep. 04, 2022. https://github.com/aquasecurity/kube-hunter

[27] T. Swimmer. “Datreeio/Datree: About Prevent Kubernetes Misconfigurations from Reaching Production. From Code to Cloud, Datree Provides an E2E Policy Enforcement Solution to Run Automatic Checks for Rule Violations.” GitHub. Oct. 04, 2021. https://github.com/datreeio/datree

[28] G. Wood. “mozilla/http-observatory-website: Mozilla Observatory (Website).” GitHub. Nov. 07, 2022. https://github.com/mozilla/http-observatory-website

[29] Z. Ghanbari, Y. Rahmani, H. Ghaffarian and M. H. Ahmadzadegan, "Comparative Approach to Web Application Firewalls," 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, Iran, 2015, pp. 808-812, doi: 10.1109/KBEI.2015.7436148.

[30] S. Alhomdy, F. Thabit, F. Abdulrazzak, A. Haldorai, and S. Jagtape. “The Role Of Cloud Computing Technology: A Savior to Fight The Lockdown in COVID 19

Crisis, The Benefits, Characteristics and Applications." COVID 19. Oct. 05, 2021. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8492023/

[31] I. Chua. "Real Life Examples of Web Vulnerabilities (OWASP Top 10)." OWASP Top 10. Jan. 04, 2022. https://www.horangi.com/blog/real-life-examples-of-web-vulnerabilities

[32] S. Foster. "Vulnerabilities Definition: Top 10 Software Vulnerabilities" Jul 27, 2020. https://www.perforce.com/blog/kw/common-software-vulnerabilities

[33] S. El-etriby, E. Mohamed, and H. Abdul-kader. "Modern Encryption Techniques for Cloud Computing Randomness and Performance Testing" ICCIT 2012 In Proceedings of International Conference on Communications and Information Technology (ICCIT2012), Hammamet, Tunisia (pp. 800-805)

[34] K. Duisebekova, K. Nationa, and R. Khabirov "Django as Secure Web-Framework in Practice" The Bulletin of Kazakh Academy of Transport and Communications named after M. Tynyshpayev ISSN 1609-1817. Vol. 116, No.1 (2021), pp.275-281

[35] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, "Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration," Sensors, vol. 20, no. 16, p. 4621, Aug. 2020, doi: 10.3390/s20164621.

[36] Holovaty, A., & Kaplan-Moss, J. (2009). The Definitive Guide to Django: Web Development Done Right, Second Edition (2nd. ed.). Apress, USA.

[37] Tari, Z., Yi, X., Premarathne, U. S., Bertok, P., and Khalil, I. Security and Privacy in Cloud Computing: Vision, Trends, and Challenges. IEEE Cloud Computing, 2(2):30–38, 2015.

[38] Theoharidou, M., Papanikolaou, N., Pearson, S., and Gritzalis, D. Privacy Risk, Security, Accountability in The Cloud. In 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, volume 1, pages 177–184. IEEE, 2013.

[39] A. Priyanka and S. Smruthi, "WebApplication Vulnerabilities: Exploitation and Prevention," Proceedings of the Second International Conference on Inventive Research in Computing Applications (ICIRCA-2020) IEEE Xplore Part Number: CFP20N67-ART; ISBN: 978-1-7281-5374-2

[40] A. S. Dikhit and K. Karodiya, "Result Evaluation of Field Authentication Based SQL Injection and XSS Attack Exposure," in 2017 International Conference on Information, Communication, Instrumentation and Control (ICICIC), Indore: IEEE, Aug. 2017, pp. 1–6. doi: 10.1109/ICOMICON.2017.8279148.

APPENDIX

The code developed during this research is available online in a GitHub repository: https://github.com/alialqarni2040/project.git