FMDB *Transactions on Sustainable Intelligent Networks*

Docker Based Decentralized Vulnerability Assessment with Port Scanning Powered by Artificial Intelligence

G. Chiranjeevi[1,*], R. Abhishek Reddy[2], R. Shyam[3], Swati Sah[4], Rejwan Bin Sulaiman[5]

[1,2]Department of Computer Science/Information Technology, Jain (Deemed-to-be University), Bengaluru, Karnataka, India.
[3]Department of Computer Science/Information Technology, Presidency College, Kempapura, Bengaluru, Karnataka, India.
[4]School of Engineering and Technology, Sharda University, Greater Noida, Uttar Pradesh, India.
[5]Department of Computer Science and Technology, Northumbria University, London, United Kingdom.
chiranjeevi.naidu1312@gmail.com[1], akabhiarvd2003@gmail.com[2], shyam7368@gmail.com[3], iswatisah19@gmail.com[4], rejwan.sulaiman@northumbria.ac.uk[5]

Abstract: Decentralized solutions, widely adopted across industries like banking, health- care, and logistics, face persistent security concerns from potential threats. This study introduces a novel decentralized vulnerability assessment using GPT-3, an artificial intelligence (AI) technology. Employing Dockerized containers for disinfecting environments and creating unique connections to the AI API service enhances system responsiveness. AI algorithms, specifically GPT-3, conduct comprehensive network scans to identify security flaws. Findings are securely distributed to network nodes, fortifying the system's defence. This departure from centralized control and traditional security audits marks a significant advancement in securing decentralized systems. AI-enabled real-time monitoring facilitates swift responses to security issues, reducing breach risks and aiding effective resource management. Encouraging results from controlled system analysis, focusing on GPT-3 vulnerabilities, highlight the integration of Dockerized containers for enhanced system efficiency. This work lays the foundation for further research, emphasizing the potential of decentralized systems for rigorous security assessments.

Keywords: Decentralization and Scalability; REST API; Artificial Intelligence (AI); Enumeration and Vulnerability Analysis; Peer-To-Peer; Information Security; Generative Pre-trained Transformer 3 (GPT-3); Load Balancing.

Received on: 07/06/2024, **Revised on:** 11/08/2024, **Accepted on:** 04/10/2024, **Published on:** 14/12/2024

Journal Homepage: https://www.fmdbpub.com/user/journals/details/FTSIN

DOI: https://doi.org/10.69888/FTSIN.2024.000290

Cite as: G. Chiranjeevi, R. A. Reddy, R. Shyam, S. Sah, and R. B. Sulaiman, "Docker Based Decentralized Vulnerability Assessment with Port Scanning Powered by Artificial Intelligence," *FMDB Transactions on Sustainable Intelligent Networks*., vol.1, no.4, pp. 220–241, 2024.

1. Introduction

Vulnerability analysis, which entails locating and evaluating any security flaws in a system that hostile actors could use, is a crucial component of network security and cybersecurity. Organizations can effectively allocate resources and prioritize security measures by reducing the risk of security breaches. The growing acceptance of decentralized systems has prompted the creation of specific vulnerability study methods designed for them. Decentralized systems rely on a network of nodes to

*Corresponding author.

jointly manage system maintenance without a central authority. Although this distributed architecture increases resilience and lessens reliance on a single point of failure, it also creates additional security concerns because every node's security is linked to the network.

This paper suggests utilizing decentralized artificial intelligence (AI) for vulnerability enumeration and investigation. The system comprises two main parts: a network scanner and an AI-powered vulnerability assessment module. The AI module evaluates the scan data to ascertain the network's security posture, while the network scanner scans to find potential security vulnerabilities. The AI and the API are two distinct systems housed on different infrastructures for increased efficiency. A temporary Docker instance is created for each new connection to the AI API service, and the container is ended when the session is over. There are various benefits to this system's use of AI. AI systems can identify a wide range of security vulnerabilities, quickly conduct security assessments, and instantly adapt to changes in the security environment. By doing away with the requirement for assessments by a trustworthy third party, this decentralized solution improves overall security compared to conventional centralized security assessment techniques. The GPT-3's real-time monitoring and quick reaction features enhance system security by enabling enterprises to identify and fix system vulnerabilities using threat modelling, network scanning, and penetration testing methods.

Data security is the main concern regarding cybersecurity in banking, healthcare, logistics, and other fields that require active computing for day-to-day operations. "Data is money" in the current day and age. This is a standing fact. The concerned data can be stolen or manipulated to get a lot of arterial motives full-filled. Cavaliere et al. [15] explain different scenarios of attacks done on medical infrastructure. The security risk here is enormous as, from a financial point of view, the hospital will lose people's goodwill and pay a lot in lawsuits and fines. However, many lives are at stake from an emotional and humanitarian standpoint. Every network node can receive the findings thanks to the vulnerability assessment system's decentralization and incorporation of GPT-3. The absence of security audits and centralized authority increases attack resistance. Furthermore, GPT-3's dynamic resource allocation and security initiative prioritization help organizations focus on fixing major security flaws, which helps build resilient, secure, and attack-proof decentralized systems. The reason for using GPT-3 is that it is an accessible and widely known version of GPT. Even though the accuracy is not on par with GPT-4, we can use this system as a base for further improvement by fine-tuning when we have a more stable version of GPT-4 we can use that has our base AI. Other AI models, such as Falcon and LLama2, are also used, but fine-tuning the model is necessary for the most part. Our paper only focuses on introducing AI and its usage, and the actual AI and its performance analysis are for future improvements.

2. Literature Review

Barrere et al. [1] explored that beginning with a definition of the terms " autonomic networks" and "services," the author of the paper goes on to discuss the significance of vulnerability assessment in these systems. After that, a thorough overview of current vulnerability assessment methods is given, including static and dynamic analysis, penetration testing, and methods based on machine learning. The difficulties and limitations of current vulnerability assessment methodologies are also covered in the article, including the time and money needed for testing, the necessity for correct and current information about the system being tested, and the problem of detecting all potential vulnerabilities.

Kertysova [2] investigates the effects of AI on misinformation, including the spread of misleading information, the swaying of public opinion, and the decline in confidence in authorities and the media. The authors contend that these consequences have broad ramifications for democracy, public dialogue, and societal operations. The article also gives a summary of current counter-disinformation measures, including media literacy campaigns, fact-checking programs, and the creation of tools to identify and eliminate incorrect information. The authors contend that while these efforts are important, they might not be enough to combat the volume and sophistication of false information produced by AI.

Harp et al. [3] detail how AI planning can automate vulnerability analysis. They clarify that a knowledge-based method of identifying potential vulnerabilities and evaluating the risk caused by these vulnerabilities can be used in AI planning to assist in automating the process. The paper overviews the AI planning systems and vulnerability analysis methodologies and discusses their difficulties and shortcomings. Compared to conventional methods, the authors contend that AI planning can offer a more effective and efficient way of undertaking vulnerability analysis.

Malkawi et al. [4] suggest automating the active reconnaissance step using an API-based port and vulnerability scanner. The program leverages APIs to obtain data about the target system, which is then analyzed by AI algorithms to find any potential flaws. The tool provides thorough information about the vulnerabilities it identifies and is made to be quick, effective, and accurate. The tool's architecture and the algorithms employed for vulnerability analysis are described in detail by the authors as other technical aspects of the tool. They also thoroughly assess the instrument, focusing on its precision, speed, and scalability. The findings demonstrate that the tool is faster and more accurate than existing methods at spotting vulnerabilities.

Papineni et al. [5] propose a method to address the limitations of extensive and expensive human evaluations. The paper suggests a quick, cost-effective, and language-independent automatic machine translation evaluation approach. This method correlates highly with human evaluation while incurring minimal marginal costs per run. Positioned as an automated understudy to skilled human judges, the proposed approach is a substitute for quick or frequent evaluations, providing an efficient alternative to time-consuming and resource-intensive human assessments in machine translation.

Frey et al. [6] contend that centralized servers and client-server architectures used in conventional virtual environments have several drawbacks, including scalability, security, and availability. They suggest Solipsis, a decentralized virtual environment design that disperses the virtual environment across several nodes via a peer-to-peer network. The primary contributions of the research are briefly summarised in the paper's abstract. Solipsis offers a decentralized architecture for virtual environments to solve the shortcomings of conventional virtual environments. The authors describe how Solipsis' decentralized architecture enables it to accomplish its main objectives: scalability, security, and availability. The paper's conclusion highlights the potential advantages of Solipsis and summarises the key research findings. In comparison to conventional virtual environments, the authors contend that Solipsis offers a virtual environment that is more scalable, secure, and accessible. Additionally, they contend that Solipsis has the potential to impact the future of virtual environments substantially and has a variety of applications, including social networks and gaming, as well as business and scientific simulations.

Abhishek et al. [7] suggested that the approach entails analyzing network traffic and spotting potential security flaws using machine learning strategies and network scanning technologies. The system's automatic classification of network components and services is intended to help find potential network vulnerabilities. The system employs AI-based algorithms to identify potential flaws and then suggests exploits that may be utilized to exploit them. The system considers several variables, such as the kind of device or service, the software version, and the known vulnerabilities. The system also intends to automatically update its database of potential exploits as new threats appear and continuously scan the network for new vulnerabilities. This enables businesses to keep abreast of the most recent security risks and take preventative steps to safeguard their networks.

Floridi and Chiriatti [8] begin by tracing the development of artificial intelligence and natural language processing, emphasizing the creation of pre-trained models like GPT-3 that make natural language processing more rapid and accurate. After that, they summarise GPT-3's architecture and capabilities, including its capacity to produce text that resembles human speech, carry out language functions like translation and summarization, and even do jobs like writing code or creating music. Floridi and Chiriatti also point out several of GPT-3's drawbacks, such as how much data it needs to operate well, its propensity for "semantic drift," its lack of context sensitivity, and its susceptibility to bias and manipulation.

Sun et al. [9] explain that within the NLP field, the paper tackles the problem of text incoherence in conditional story generation and contextual text continuation. Cur- rent models frequently have the problem of progressively deviating from the prompt, leading to writing that looks nonsense to humans, even with reasonable complexity and variety. The authors suggest a two-step process to improve language generation's consistency and coherence. First, they use a GPT-2 pre-trained model to build a sentence pair coherence classifier. The GPT-2 language model is then co-trained with a new coherence objective by an approach similar to the REINFORCE algorithm. The resulting refined model reduces the issue of divergence by producing long paragraphs conditioned on a certain subject.

Chiranjeevi et al. [13] said that this is a project our team came up with to generate the training models necessary to have an initial training. The code repository comprises two training datasets created for GPT3.5-4 and LLama 7b models. In [14], this is a project our team came up with to create a proof of concept of how effective GPT models and LLama models can be in doing vulnerability analysis on network scans, DNS scans, and PCAP analysis. Hu et al. [10] dive deep into how the LORA LLM Training technique works and how it can be used to train the GPT3.5 and other Large Language Models.

Sklansky et al. [11] offer a novel method for computer-assisted nonlinear classifier and cluster finder discovery suited for complex multidimensional data distributions. Their methodology is based on the coordinated operation of three main components: Trainable Linear Arrays (TLAs), Feature Expanders (FEs), and Feature Selectors (FSs). FSs use advanced selection techniques like genetic and relaxed branch-and-bound selectors to extract relevant features. FEs create an advanced feature set from a primitive one by mating and mutating using genetic algorithms and algebraic operations. Using a set of trainable linear weights, TLAs transform features even more to enhance their quality. Combining these methods—hill climbing, genetic search, and algebraic transformations—leads to a parallel nonlinear classifier that can identify intricate data distributions. Singh and Singh [12] go deep into the key differences between GPT3 and GPT4 models and explain the core mechanics behind these differences and what makes these changes vital. The paper aims to uncover various core aspects, such as similarities, performance, and accuracy improvements, to name a few.

Cavaliere et al. [15] explain the impact and extent of security attacks on hospitals and other medical infrastructure. The efficiency of the suggested model in generating expanded text that is in line with the prompt sentence's main idea is

demonstrated by the examination of the model. The co-trained model shows superior subject adherence than pure language models, such as GPT-2, which tend to stray from the topic over time. Several experiments demonstrate the suggested method's importance, including linear and nonlinear changes in prompt sentence embeddings. Compared to the pure GPT-2 language model, the generated text sometimes has less natural wording and syntax, although it retains topic coherence. However, the scientists point out that there is a trade-off between the model's capacity to match each generated word to a specific topic, and they call for more effort to improve fluency without sacrificing coherence.

3. System and Usage

AI hosting and API hosting are the two main parts of the system. These parts interact with each other even though they are hosted in different settings with different capabilities since they communicate seamlessly using APIs. This method guards against resource misuse and guarantees error-free communication. Due to their distinct features and capabilities, the AI and API hosting systems require an environment-appropriate communication protocol. Dockerized containers are essential for improving productivity and resource management. AI and API hosting can be done in isolated environments with consistent resource usage thanks to Dockerization. Using Docker containers avoids performance bottlenecks and resource conflicts as each component runs within its allotted resources. The streamlined and optimized system operation is further enhanced by Docker's resource management capabilities, which facilitate the effective allocation and utilization of computing resources.

To further ensure resource consistency, Docker's garbage collection mechanisms are essential. Docker eliminates clutter and maintains optimal performance in the system by automatically detecting and eliminating unused or unnecessary resources within the containers. This is especially helpful in decentralized systems because it reduces the possibility of wasting resources and encourages the more sustainable use of computing power. With Docker's resource management and garbage collection capabilities, the system's decentralized architecture adds more security while guaranteeing safe communication between AI and API components. The lack of a central data repository reduces the system's susceptibility to potential hacker attacks and security lapses, improving its overall resilience. Additionally, vulnerability analysis can proceed smoothly using a decentralized and Dockerized approach, facilitating easy collaboration between the AI and API while preserving optimal resource utilization and system efficiency.

3.1. Decentralized System

Because each node in a decentralized system has equal power and responsibility, it is more secure and resilient than a centralized system because it has a peer-to-peer network structure without centralized control. Artificial intelligence (AI) and decentralized systems have been the subject of recent research. This entails hosting an AI system and a RestAPI inside the decentralized architecture, two essential components. The AI system is responsible for data analysis and prediction, while the RestAPI system manages incoming requests from external systems. One approach to hosting these systems is through a decentralized virtual machine (DVM) system, as detailed in [6]. In a DVM system, the virtual machine is distributed across the network, with each node hosting a segment, facilitating processing and memory contributions. This collaborative node-based approach ensures decentralized operation, enhancing resilience against errors and attacks.

The decentralized system proved essential for AI applications, which frequently handle sensitive data. By dispersing data storage and computations throughout the network and removing a single point of vulnerability, this design guards against data breaches. Furthermore, safe and dependable data sharing is ensured by using APIs to communicate between the RestAPI and AI systems, an essential feature for managing sensitive data. The decentralized design offers flexibility and scalability because the RestAPI system can run on a smaller server, and the AI system can be hosted on a powerful server with plenty of resources. Because both systems can be hosted in different locations, this design improves system redundancy and resilience by lessening the impact of failures or outages. The usage of APIs to enable safe and dependable communication between the RestAPI and AI systems is a crucial component of this setup. APIs facilitate the construction of complex and sophisticated systems by enabling integration with other apps and systems and guaranteeing the integrity of data interchange. Therefore, this method offers a complete and reliable way to host and integrate RestAPI and AI systems in decentralized contexts.

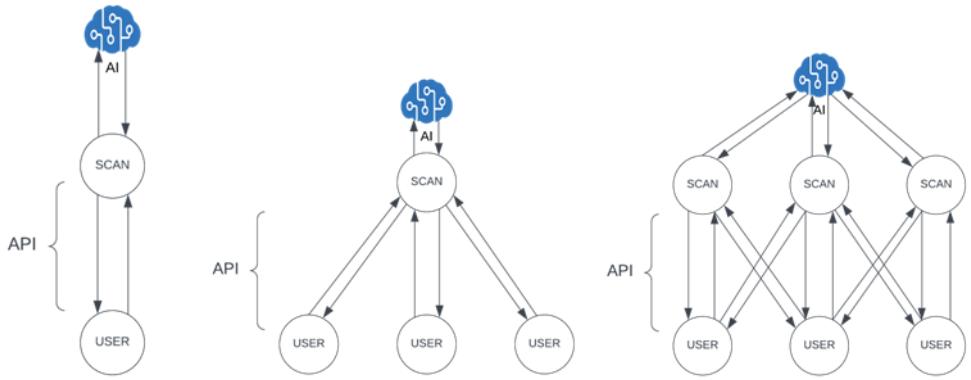
3.2. Methods of Applications

We can have three main approaches here they can be:

- **One-to-one:** one docker instance for one user.
- **One-to-many:** one docker instance for two or more users.
- **Many-to-many:** There are many docker instances for many users.

One-to-one: 1a, In this form of deployment, we will have one scanner docker instance and one user. For every new user, one scan instance can be deployed and maintained. The usage of this instance can change. The instance can be created for a session, and once the session ends or the time delay between the instances is greater, the instance can be destroyed. This system comes with its set of flaws or downsides. The downside is that the system is only accessible by one person, leading to over or under-usage. The system is also not running at its fullest at all times, leading to wasted resources. Therefore, this method of usage is unreliable.

One-to-many: 1b. In this form of deployment, we will have one scanner instance serving many user requests. This immediately rules out the resource utilization and wastefulness issue from the one-to-one method, But this has several more flaws under the hood. The major problem here is the usage of resources and lack of redundancy. If any issues occur, the service won't be capable of surviving such a high load. This will also affect how this paper's garbage collection and load management implementation will be used (Figure 1).



One to One deployment One to Many deployments Many to Many deployments

Figure 1: Combined Deployment

Many-to-many: 1c, So far, this is the best method of deployment is the best for our case as implementing such a large-scale deployment is fees-able and can also have much more room for redundancy and error handling, not to mention the usage of the garbage collection and load balancing will also play a vital role in maintaining overall resourcefulness and efficiency. For this paper, we have analyzed how the scanner API, OpenAI API, and the end-user interface work together, but we have not done a large-scale implementation. As for that, the many-to-many approach is considered but on a much smaller and comparable scale. The dockerized containers are a way of implementing a resource manager and usable shell for a temporary instance. The container maintains a clean environment without background noise or disturbance in the scanner's performance. The containers act as a perfect incubator for the scanner's operation. The method of implementing the scanner within the dockerized container is as follows:

- **Develop the scanner**: Make it so that it can communicate both in and out with the AI API system and the control system for the scanner.
- **Controls:** The System must be equipped with all the necessary logic to handle one or many connections.

The methodology of implementation we followed is as in 2: The internals of the docker-sized system contain the profile checker, the main scanner, some amount of temporary storage for the AI and scan data, the container to AI API connector, and finally, the data conversion and regex. Each component has a specific task and is controlled by sequential logic. The AI API connection is also important as it hosts the second most important component of the entire system, the AI. For this purpose, any cloud hosting can be used if, in a professional environment, a second EC2 instance, such as AWS or a specialized runpod headless server, which is specifically used for AI hosting, can be used. The communication between these channels can be done on HTTPS using custom TLS certificates for enhanced security.

3.3. Scanner API

Identifying security flaws or vulnerabilities in a computer network or system, commonly known as vulnerability scanning, is a pivotal aspect of cyber security. Often conducted using automated tools or software, vulnerability scanning thoroughly examines a system's security posture, searching for potential vulnerabilities, incorrect setups, or other weak points. The objective is to provide enterprises with a comprehensive awareness of their security risks, enabling them to take necessary steps

to mitigate or eliminate potential threats. Developers can programmatically access the features of the well-known open-source network scanner, Nmap, through the Nmap API. This API provides a straightforward and adaptable approach, allowing Nmap scans to integrate into unique software, scripts, or automation workflows. Supporting various output formats and scan types and offering extensive customization and configuration options, the Nmap API enables developers to quickly and easily identify hosts and services on a network, along with crucial details about their security posture.

In the context of this research [1], challenges associated with vulnerability assessments in autonomous networks are highlighted. These challenges include the size and complexity of networks, the dynamic and ever-changing nature of their components, and the imperative to maintain network security during evaluations. The study recommends further research and development, offering a critical review of the advantages and disadvantages of various vulnerability assessment approaches. The researcher explores various scans and optimal techniques to enhance output quality. Another related study [4] delves into the use of API and automation in the active reconnaissance phase of vulnerability assessment. The proposed system combines API- based port scanning with automated vulnerability scanning to enhance the efficiency and accuracy of the reconnaissance process. The paper outlines the system design, implementation, and evaluation results, demonstrating optimal performance during scans on targeted systems. Based on this research, there is potential for building a REST API for Nmap.

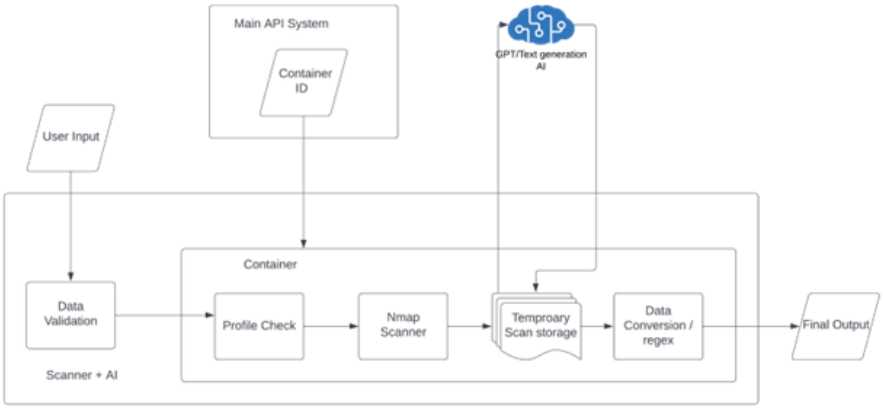


Figure 2: Docker Instance Internal

The current study introduces a session-based approach for vulnerability scanning using temporary Docker images, enhancing flexibility and security. An endpoint API call must be made for scanner image generation to initiate a scan. The specified image name from the generation process must be mentioned during each scan request. At the end of the usage session, an API call for destruction is essential, ensuring the termination of the temporary Docker image. Additionally, the exploration of fog computing for a similar process is acknowledged, presenting the possibility of leveraging distributed resources for improved efficiency and scalability in vulnerability scanning within a decentralized environment. Figure 2 shows a simple overview of how the system will be set up internally within a docker image.

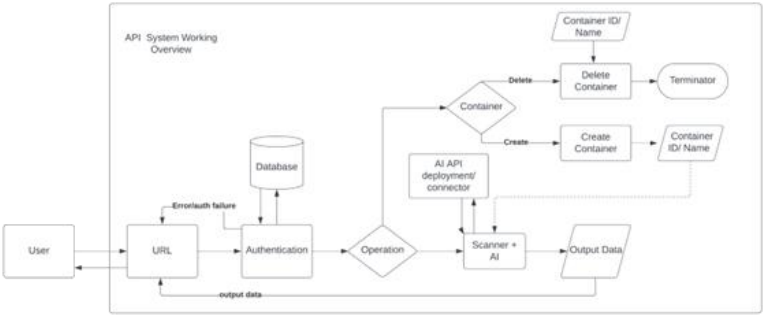


Figure 3: API Overview

As seen in Figure 3, the API contains a serialization feature that controls the incoming and outgoing results of the queries and distributes the result data to the intended user accordingly. When paired with other API features like steady scans and converters and scanning techniques for trustworthy output, these capabilities can result in an incredibly accurate scanner system. Figure 4

shows a simple overview of the system's setup. As seen in Figure 4, The API first conducts its authentication, and based on the call, the API decides on either the generation of a new image or the removal of the image. On a much larger implementation, this is a crucial segment that conducts the generation, scanning, and destruction part of the docker instance. The container or the scanner can be a separate system entirely and connected via API or any other means of connection. Once the API request is made, the process is as such, and the next step is authentication. Once the authentication has been completed, there are two modes available:

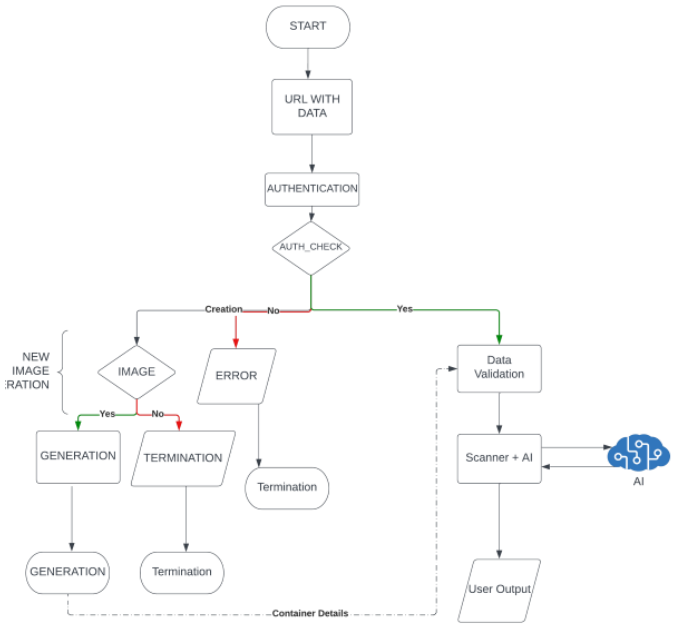


Figure 4: Overall System operational overview

- **Container:** This has two modes: 1 for creation and the second for deletion. The container is created after a few checks. For now, these checks are not needed for a small-scale deployment.
- **Scanner:** This mode takes the container ID from the creation, and using that, the scans are conducted and tracked. The container ID is used to run the container, and once the scan has been completed, the output is then collected and shared as the final output.

3.4. Load Balancing and Garbage Collection

We must also consider the node's capabilities and decide how many systems can be generated to implement any garbage collection system or functionality.

$$Load_{total\%} = \left(\frac{CPU_{use} \times I}{1000} + \frac{RAM_{dock} \times I}{RAM_{tot} \times 1024} \right) \times 100$$

- Loadtotal%: Represents the total load in percentage.
- CPU use: This is the CPU usage percentage for one instance.
- I: Is the total number of instances.
- RAMdock: This is the RAM usage in MB for one instance.
- RAMtot: This is the total available RAM in GB multiplied by 1024 to convert to MB for consistency with RAMdock.

The formula for determining the total load percentage of Docker instances on a server is designed to assess the cumulative impact of CPU and RAM utilization by Docker workloads. It ensures that the server's resources are not overcommitted, crucial for maintaining system stability and performance. By calculating the resource usage as a percentage of the total available, the formula provides a scalable metric that can be applied regardless of the server's size or capacity. The resulting load percentage is pivotal for determining the feasible number of Docker instances that can be concurrently deployed and effectively managed. This capacity planning is essential for both short-term performance and long-term resource management, ensuring that each

Docker instance has adequate resources to function optimally without affecting the others. For optimized garbage collection and system maintenance, the following utilization thresholds are established:

- **MAX CPU:** This threshold limits the CPU usage to prevent overloading. It is essential to ensure that Docker instances do not consume CPU resources beyond a critical point, thereby maintaining system responsiveness and stability.
- **MAX RAM:** Similar to the CPU limit, this threshold caps RAM usage by Docker instances. By keeping memory consumption under this limit, the system can avoid excessive swapping and maintain adequate free memory for smooth operation.
- **MAX LOAD:** This threshold is determined by a calculated load metric, reflecting the overall resource utilization. Staying below this limit ensures the system can accommodate fluctuating workloads without risking resource exhaustion or performance degradation.
- **GC TIMEOUT:** Defines a periodic interval for evaluating system load and resource utilization, triggering garbage collection as necessary. This proactive measure helps in reclaiming unused resources and preventing potential bottlenecks.

Implementing these thresholds requires a systematic approach to monitoring and managing resource utilization:

- **Resource Metrics Collection:** Gather current CPU and RAM usage statistics to understand the immediate resource consumption landscape.
- **Load Calculation:** Utilize a predefined formula to assess the total load on the system, incorporating both CPU and RAM usage metrics for a comprehensive overview.
- **Threshold Evaluation:** Compare current resource metrics against the set thresholds to identify conditions that necessitate garbage collection.
- **Garbage Collection Activation:** Trigger garbage collection processes when any threshold is breached or at regular intervals, as the GC TIMEOUT dictates. This ensures the system remains within operational parameters, preventing overutilization and enhancing overall performance. Adhering to these guidelines forms the cornerstone of effective resource management, guaranteeing that Docker instances operate efficiently without adversely affecting the system's health or performance.

The Garbage collection logic checks for the values of at least 2 of the values exceeding the threshold values or if the TIMEOUT has been met. The garbage collection can be done in 2 ways.

- METHOD 1: Reboot
- METHOD 2: Re-create

In method 1, we can reboot each of the instances, and by doing this, we can easily reduce. The resource utilization is not significant, will not put the entire system on halt, and is more reliable. In method 2, we can delete, re-create, and host the instances. This method is completed for implementation and can become rather hard to implement over a large scale. However, this system helps reduce resource usage and increases efficiency. The logic behind the garbage collection looks like this:

Listing 1: Optimized Load-Based GC Methodology

```
# Resource Utilization Thresholds
MAX_CPU, MAX_RAM, MAX_LOAD, GC_TIMEOUT = 90, 75, 80, 60

# Garbage Collection Triggers
cpuOverload = ramOverload = loadExcess = False

# Collect Resource Metrics
currentCpu, currentRam = get_cpu_ram_usage()
totalLoad = calc_load(currentCpu, currentRam)

# Check Against Thresholds
cpuOverload = currentCpu >= MAX_CPU
ramOverload = currentRam >= MAX_RAM
loadExcess = totalLoad >= MAX_LOAD

# Trigger Garbage Collection
if time_since_last_gc >= GC_TIMEOUT or cpuOverload or
    ramOverload or loadExcess:
run_gc()
```

3.4.1. Purpose of Load Formulation

The load formulation is instrumental in ensuring the efficacy of garbage collection methodologies within Docker environments. By calculating the load, we can pre-emptively manage and mitigate resource exhaustion, thereby maintaining system stability

and performance. Incorporating MAX CPU and MAX RAM thresholds into our calculation is crucial; however, their value extends beyond mere limits. The collective resource usage across all running instances must be carefully managed to prevent resource allocation issues. This is vital to avoid performance degradation or system burnout due to over-allocation or under-utilization of resources. To enhance the precision of our load calculations, it is imperative to define MAX CPU and MAX RAM at an instance level and across the entire Docker ecosystem. This holistic approach ensures that the cumulative resource usage remains within acceptable bounds, thus facilitating the smooth running of all instances without impeding the allocation of resources to new or existing instances. Furthermore, this methodology allows for dynamic scalability and the continuous improvement of the Docker environment. By monitoring and adjusting the load thresholds based on real-time usage and performance metrics, administrators can optimize resource utilization, ensuring that the system can scale effectively to meet demand while maintaining optimal performance.

Scalability Advantage: The dynamic nature of load-based resource management significantly enhances the scalability of Docker environments. By adapting to changes in workload in real-time, resources can be allocated or freed up as needed, ensuring that each container has access to the necessary resources without wasting capacity. This adaptability is crucial for environments that experience variable workloads, enabling seamless scaling up or down to match demand. Scalability is further enhanced by the ability to automate the deployment and shutdown of instances based on load conditions, making it possible to maintain a balance between performance, cost, and resource efficiency. This approach reduces the likelihood of performance bottlenecks and ensures that resource utilization is optimized across the board, laying a strong foundation for scalable cloud-native applications.

3.4.2. Considerations for Calculating Load Percentages in Docker Environments

- **CPU Utilization Metrics:** It's crucial to account for the actual CPU usage reported by Docker, which can be accessed via Docker stats. These metrics provide insights into the total utilization of each container relative to the resources allocated to it. The utilization reflects how effectively a container uses the CPU cores assigned to it, which is essential for accurate load calculation.
- **RAM Usage Metrics:** Similar to CPU metrics, RAM usage by each Docker instance should be considered based on actual measurements rather than allocated limits. Docker stats also offer this information, showing how much memory each container uses in real time. This is important because RAM consumption directly impacts the load calculation and system performance.
- **Dynamic Resource Allocation:** The load calculation should dynamically reflect the number of cores and total RAM allocated to the Docker environment. This approach ensures that the load percentage accounts for real-time resource usage rather than relying on static or predetermined allocations. It allows for more flexible and responsive system management.
- **Simplified Load Formulation:** If a predetermined percentage of resource usage is assumed, the load calculation can be simplified to the following formula:

$$Load_{total\%} = \frac{(CPU_{use\%} + RAM_{use\%})}{2}$$

This formula averages the CPU and RAM usage percentages to derive an overall load percentage. However, this approach may not accurately reflect the system's actual state unless real-time usage metrics are used.

3.4.3. Applications of Load Formulation Methodology

The load formulation methodology, designed for Docker environments, applies across various computing paradigms. This methodology's adaptability and efficiency in managing resources dynamically make it suitable for various scenarios:

- **Cloud Computing:** The methodology can potentially optimize resource allocation across multiple virtual machines and containers in cloud environments. Cloud service providers can lower operating costs and increase user satisfaction by managing multi-tenant environments more skillfully and providing higher service levels by ensuring that resources are used efficiently.
- **Edge Computing:** Dynamic load management is especially useful in edge computing scenarios, where computing resources are closer to the data source. By dynamically allocating resources according to workload, it facilitates real-time data processing and analysis with minimal latency, ultimately improving the performance of Internet of Things applications and services.
- **Fog Computing:** The idea of edge computing is expanded upon by fog computing, which offers a decentralized computing infrastructure. In this case, the load formation methodology is especially helpful because it makes it possible to distribute computing resources across the fog layer effectively. This guarantees distributed applications

operate at their best, particularly when prompt decision-making and processing near end users or data sources are necessary.

- **Human-Machine Networks:** The approach assists with load balancing between private and public clouds in hybrid cloud configurations where workloads are split between them. It guarantees that applications operate without hiccups by dynamically adjusting resources and taking advantage of the best options available regarding cost, performance, and compliance with regulations.
- **Architectures for Microservices:** This approach makes it easier to scale individual microservices according to their respective loads in applications organized as a collection of loosely coupled services. The overall effectiveness and responsiveness of microservices-based applications depend on this fine-grained control over resource allocation.
- **HPC (High-Performance Computing):** In high-performance computing (HPC) environments, load formulation methodology is crucial in dynamically allocating CPU, RAM, and other resources to computational tasks. Computation speed and efficiency are increased by ensuring that high-demand tasks receive timely resources. The load formulation approach improves system performance, scalability, and resource utilization in these diverse computing environments. The approach to resource management that it offers is dynamic and responsive, catering to the various requirements of contemporary computing paradigms such as cloud and fog computing, as well as HPC and microservices architectures.

3.5. Vulnerability Assessment AI

Applying machine learning techniques to enhance network security in areas like intrusion detection, vulnerability scanning, and exploit prediction has become popular in cybersecurity. Abhishek et al. [7] suggested a similar approach to implementing a vulnerability analysis system. An automated machine learning-based method to analyze the Nmap API JSON output and identify vulnerabilities with proposed exploits can be helpful in this situation. This is possible via supervised learning, in which the AI is trained on a sizable dataset of prior Nmap scan results and known vulnerabilities, with the output labels being the projected flaws and suggested exploits.

The Python library Nmap-Python, sometimes called the Nmap API, is used to conduct network scans and compile data about network hosts. A target system can be scanned using the Nmap API to learn about its open ports, currently active services, and operating system specifics. The AI algorithm will utilize this information to forecast the system vulnerabilities when serialized and supplied. A sizable and varied dataset is crucial for training the AI algorithm's predictions to increase accuracy. This dataset ought to include a wide range of various systems, setups, and vulnerabilities. It should be continuously updated with fresh data and retrained to guarantee that the AI system effectively identifies new and evolving vulnerabilities. The accuracy and speed of the vulnerability analysis process could be significantly increased by using an AI algorithm to automate the procedure. Security experts can lighten their workload and improve system security by using the Nmap API to gather data on a target system and a machine learning-based algorithm to forecast the present vulnerabilities. To ensure the AI algorithm's continuous efficacy, it is crucial to assess the quality of the training data carefully and to update and retrain it frequently.

Kertysova [2] discusses AI disinformation and its effects. Artificial intelligence and disinformation are terms used to describe the dissemination of incorrect information and the use of AI to sway public opinion. Artificial intelligence (AI) algorithms that propagate misleading information have the potential to influence political processes and decision-making and harm both individuals and society. The danger of deepfakes, which can be used to disseminate misinformation or even extort individuals, has surfaced with AI algorithms that produce convincing movies and audio. AI may produce highly convincing, bogus content that is challenging to tell apart from the genuine thing, making it challenging to combat AI-generated disinformation. Some preventive methods can be taken against these issues, such as more intense training that uses all possible aspects of a scenario.

Harp et al. [3] discuss how Automated Vulnerability Analysis is used to study and evaluate system vulnerabilities through artificial intelligence approaches used in planning. This research aims to automate the manual effort and expertise-intensive process of finding and evaluating system flaws. The vulnerability analysis is carried out using plans or sequences of actions that are generated by AI planning. This strategy allows the system to consider the effects of several options and choose the best one depending on the system's present condition and the desired results. The vulnerability analysis process will be more effective, take less time and effort, and consistently produce better results.

Feature Selectors (FSs), Feature Expanders (FEs), and Trainable Linear Arrays (TLAs) are used by the author in [11] as essential parts of a carefully crafted network. These elements are essential to the computer-assisted identification of cluster locaters and nonlinear classifiers for intricate, multidimensional data. To extract pertinent features from the multidimensional data, the author uses advanced techniques like genetic and relaxed branch-and-bound selectors within the feature selectors. Feature expanders take a basic set of features and turn them into a more complex set by performing algebraic operations, mating, and mutation processes from genetic algorithms. Finally, Trainable Linear Arrays transform one set into another by applying trainable linear weights to refine features. A parallel nonlinear classifier combines hill climbing, genetic search, and

algebraic transformations. This all-inclusive approach tackles problems caused by intricate data distributions, enabling effective feature extraction for machine learning uses.

One popular metric for evaluating the quality of machine-generated text is the BLEU (Bilingual Evaluation Understudy) evaluation technique described in [5]. This technique is especially useful when evaluating machine translation and other natural language processing tasks. BLEU was initially created to assess the accuracy of machine-generated translations concerning human language, but it has since been used for various text-generation tasks. By calculating the overlap of n-grams—consecutive sequences of n words—between a machine-generated text and one or more reference texts, BLEU assesses the degree of similarity between them. The fundamental idea is to compare the n-grams in the machine-generated text to those in the reference texts to calculate a precision score. The following is the formula for BLEU:

$$BLEU = BP \times \exp\left(\sum_{n=1}^{N} w_n \cdot \log(p_n)\right)$$

Where: - BP is the brevity penalty to address the issue of shorter machine-generated texts receiving higher precision. - N is the maximum n-gram order considered. - wn is the weight assigned to the precision of n-grams. - pn is the precision of n-grams. The precision of n-grams is calculated by dividing the number of n-grams in the machine-generated text that appear in the reference texts by the total number of n-grams in the machine-generated text. The BLEU score, ranging from 0 to 1, is a comprehensive indicator of the quality of machine-generated text, with 1 representing a perfect match with reference texts. Widely employed in natural language processing tasks, the BLEU metric provides a balanced evaluation by considering precision and recall. A high BLEU score signifies that the machine-generated text aligns closely with the reference texts, indicating strong precision and high recall. Effectively integrating these aspects into a single metric, BLEU offers a nuanced assessment of text generation model performance. Suitable algorithms for achieving high BLEU scores include Advanced Neural Networks (ANNs) and Large Language Generative Models, leveraging complex neural architectures for enhanced language pattern capture.

Advanced artificial intelligence models called text generative models, such as LLAMA and OpenAI's GPT, particularly GPT-3, are made for natural language processing tasks. Facebook AI Research's (FAIR) LLAMA is incredibly versatile, with its transformer-based design enabling it to be pre-trained on a wide range of datasets before being applied to various scenarios. Similarly, OpenAI's most recent version, GPT-3, is a massive model with 175 billion parameters pre-trained on a tremendous amount of text data from the internet. These models are very good at interpreting context and identifying complex relationships in text. Their scalability enables various applications, such as summarization, translation, question answering, and even programming jobs. They function through unsupervised learning. Their fine-tuning capabilities allow them to be tailored to specific objectives, making them effective tools for generating natural language in various contexts. Ongoing research focuses on refining these models for better contextual subtleties, decreased biases, and enhanced efficiency. The advanced training systems can also utilize LORA or Low-Rank Adaptation of Large Language Models to train the data quantized for better accuracy [10]. The authors have gone in-depth into how the training works and how to implement such a complex method for accurate results.

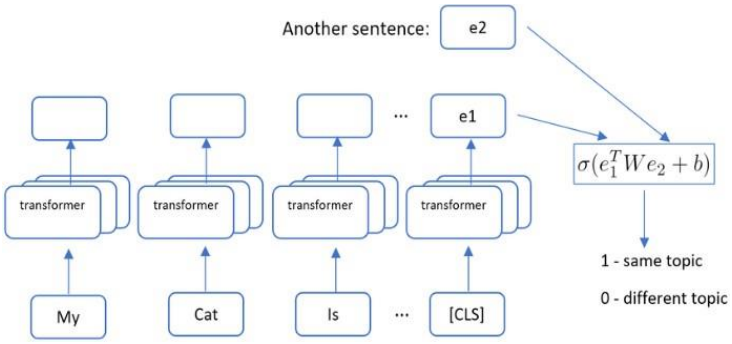


Figure 5: AI decision tree

According to the Nmap-API analysis, the input for the tree in Figure 5 from [9] Large language models (LLMs) and their underlying architecture, known as transformers, have transformed natural language processing by effectively capturing complex contextual relationships seen in textual data. Transformers use an internal process known as self-attention. The transformer architecture in LLMs, like OpenAI's GPT or BERT, comprises layers of self-attention processes that enable the model to assess the relative weights of various words in a phrase. Fundamentally different from conventional sequential models, a transformer handles input sequences in parallel. Self-attention, in which each word in a sequence pays attention to every other word and weights them according to their contextual significance, is how this parallelization is accomplished. This allows the

model to consider a word's whole context, leading to a deeper comprehension of language semantics. LLM Transformers are usually pre-trained on large text corpora, where they learn to comprehend contextual links between words or anticipate the next word in a sequence. The model gains a wide understanding of language patterns and contextual nuances during this pre-training phase. LLMs adjust to domain-specific data when fine-tuning certain tasks, making them adaptable for various applications.

3.5.1. Fine Tuning

In the case of pre-trained models, fine-tuning is an especially important stage in the machine learning model training process. Fine-tuning extends a model's expertise for particular tasks or domains once the model, typically a convolutional neural network (CNN) or large language model (LLM), has been pre-trained on a large dataset for a general comprehension of patterns and features. The model is exposed to a more limited dataset pertinent to the intended application during fine-tuning, which modifies its parameters to better suit the work's complexities. Through this process, the model can become more specialized and adaptive, catching subtleties unique to a certain domain and enhancing performance on the target job, be it picture recognition, text classification, or something else entirely. LLAMA fine-tuning entails customizing the previously trained language model to focus on cybersecurity-related tasks. LLAMA usually uses examples of instructions, input sequences, and matching desired outputs in its training data format. For example, in the data format that is provided:

Listing 2: LLAMA CVE Example

```
{
        "instruction": "Explain CVE-1999-0001",
        "input": "Explain the vulnerability: CVE-1999-0001",
        "output": "ip_input.c in BSD-derived TCP/IP ... via
            crafted packets.",
        "Affected Products": [
        {
                "tags": ["x_refsource_CONFIRM"],
                "url": "http://www.openbsd.org/errata23.html#
                    tcpfix"
        },
        {
                "name": "5707",
                "tags": ["vdb-entry", "x_refsource_OSVDB"],
                "url": "http://www.osvdb.org/5707"
        }
        ],

"CVE State": "PUBLISHED "
}
```

In listing 2, the "instruction" gives the task instructions; the "input" is the information needed for the task; and the "output" is the anticipated answer or justification. In this instance, a specific Common Vulnerabilities and Exposures (CVE) identifier is linked to a vulnerability the model has been trained to explain. To train LLAMA for cybersecurity, a variety of datasets with comparable cases that cover a range of cybersecurity themes, vulnerabilities, and their justifications must be presented to the model. To produce logical and pertinent replies to the context, the model must first learn to comprehend the contextual relationships between the various components of the input data. LLAMA can understand language subtleties thanks to its pre-training on a generic language model, and its comprehension of cybersecurity is further enhanced by fine-tuning.

LLAMA fine-tunes its parameters by considering the unique structures and patterns in the cybersecurity training data. When the model is presented with new cybersecurity-related tasks, it is prepared to offer perceptive and contextually relevant answers. This method takes advantage of the strength of pre-trained language models and adapts them to meet the unique demands of cybersecurity scenarios. The supplied data snippet is an example of OpenAI's fine-tuning format, which has a conversational framework with role-based messages.

In Listing 3, a system notification ("CVE Vulnerability Information") provides background and starts the conversation. After that, the user gives a command, and the assistant complies. Models can comprehend the conversation's flow thanks to this format, which enables them to provide context-aware responses. The training dataset for this kind of interaction would include many of these exchanges, encompassing a range of vulnerabilities, user inquiries, and assistant replies, to fine-tune a model for cybersecurity duties. The model learns to produce relevant and educational responses based on the conversational environment and its place in the discourse. The system message directs the model's comprehension of the current task by establishing the general context for the discussion. In [13], we created a reference dataset for this training, and all the example codes were referenced there. The dataset is meant to be a reference for advanced applications. Conversation-based fine-tuning works especially well when a model must read and react to multiple turns of interaction, like when a user asks a model about cybersecurity risks, and the model responds with a lengthy explanation. This style works especially well for teaching models how to have lively, context-aware interactions. In this example:

Listing 3: OpenAI CVE Exaple

```
{
        "messages": [
        {
                "role": "system",
                "content": "CVE Vulnerability Information"
        },
        {
                "role": "user",
                "content": "Explain the vulnerability: CVE-1999-0
                    001"
        },
        {
                "role": "assistant",
                "content": "CONTENTS"
        }
        ]
}
```

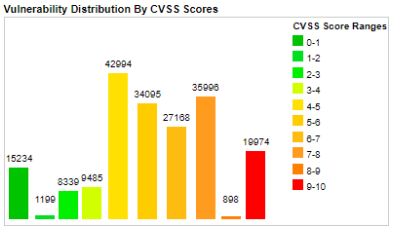
3.5.2. AI Intended Output

When an AI checks service for vulnerabilities, it primarily looks for the CVE number, the vulnerability number, and the service's version. Each of them will be compared to the database of vulnerability-related exploits, which has a sizable list of exploits from various reliable sources. The objects or each port scanned from the JSON input will be categorized after the information is acquired, and a report will be generated depending on whether an exploit was discovered or not. When the AI has finished its task, it compiles the data in its JSON and delivers it back to the Nmap-API to give to the end user, as depicted in Figure 2. The AI will learn and select the scan's outcome based on the vulnerability score. We can examine how many vulnerabilities are indicated for each scoring degree in Figure 6a and Fig. 6b. The scope of the AI is narrowed, and the scan proceeds more quickly.

Distribution of all vulnerabilities by CVSS Scores

CVSS Score	Number Of Vulnerabilities	Percentage
0-1	15234	7.80
1-2	1199	0.60
2-3	8339	4.30
3-4	9485	4.90
4-5	42994	22.00
5-6	34095	17.50
6-7	27168	13.90
7-8	35996	18.40
8-9	898	0.50
9-10	19974	10.20
Total	195382	

Weighted Average CVSS Score: **6.1**



(a) CVE distribution based on score (b) Chart on the intensity of vulnerability

Figure 6: Combined Figures

The AI can use real-life scenarios of various concepts and from various case studies to suggest an appropriate and reliable solution to the vulnerability. This was the case if The entire AI was to be built from the ground level. We will use the GPT3 AI model from OpenAI for the current paper simulations, which has an API that meets the study's goals and requirements. Floridi and Chiriatti [8] explain that the transformer architecture used by GPT-3 is effective at processing enormous volumes of text data. The model is one of the biggest NLP models, with 175 billion parameters. To function, GPT-3 generates text based on the input it receives. GPT-3 employs a pre-trained language model that has become adept at deciphering the context and significance of words and sentences. This pre-training enables the model to produce grammatically sound, coherent, and smoothly flowing content. Many potential uses for the model's capacity to produce text that sounds like human speech include automated content creation, chatbots, and language translation. GPT can use the OpenAI API as a language model to analyze JSON data vulnerability. GPT-3 is one of the models available in the OpenAI API that may be used for natural language processing activities like vulnerability assessments. For example:

Listing 4: GPT with Nmap Implementation

```
# Scan and select
nm.scan('{}'.format(ip), arguments='-Pn -sV -T4 -O -F')
json_data = nm.analyse_nmap_xml_scan()
analize = json_data["scan"]

# Prompt about what the quary is all about
prompt = {}

# A structure for the request
completion = openai.Completion.create(
        engine=model_engine,
        prompt=prompt,
        max_tokens=1024,
        n=1,
        stop=None,
)
```

In Listing 4, GPT would first need to parse the JSON data and extract pertinent information about the network and its security to conduct a vulnerability analysis of the data. IP addresses, server names, software versions, and network configurations are examples of this data. After the pertinent data has been retrieved, GPT can utilize the OpenAI API to provide a report outlining potential network vulnerabilities. The report can provide details on software versions with known vulnerabilities and potential attack methods based on network configurations. In the above example code, the Nmap scan is parsed into only sending a segment of the entire scan to the AI to assess the segment, which is the key part that must be scanned. Chiranjeevi et al. [14] have implemented a complete and advanced implementation. The Output of the AI is as follows:

Listing 5: GPT Analysis output

```
{
        "vulnerabilities": [
        {
                "name": "FTP",
                "type": "Open Port",
                "severity": "Medium"
        },
        {
                "name": "SSH",
                "type": "Open Port",
                "severity": "Medium"
        },
        {
                "name": "Telnet",
                "type": "Open Port",
                "severity": "High"
        },
        {
                "name": "SMTP",
                "type": "Open Port",
                "severity": "Medium"
        }
        ]
}
```

The Listing 5 Output of GPT was done on a simple vulnerable machine with less prompt regarding the necessary explanations of the exploits, references, and vulnerability description.

3.6. Connection and security

In a deployment similar to the above, a closed environment is suggested. This includes the components not being exposed to the outside internet except for the controls meant to be given to the end users or the staff systems meant for the job. Let's say we have to deploy this system in an AWS infra. For that to happen, we need to have four instances. They are:

- The interface
- The scanner
- The AI
- The database

The Interface: The interface is the interfacing system for all operations. It is connected to the scanner system that hosts all the adjacent systems within the scanner instance. It can check for the scanner system's system, Docker, and network statistics and perform all the necessary operations accordingly. This is the brain of all the logical operations. The scanner is the hosting system for all the different scanner containers. This has a fluctuating number of scanner instances and will be connected to both the interface and the AI systems. The major aspects of this system are sending systems statistics, scan results, and error reports to the interface while felicitating the AI information feed for all the scanner instances. This system also receives commands for the garbage collection system's operation. This system also manages each request sent by the docker instance to the appropriate

AI requests and maps them accordingly. This is the largest among the three systems and is focused on more storage, RAM, and CPU to host several docker instances.

The AI: This system hosts the AI. This fairly large and GPU-intensive system has the most resources allocated to AI tasks. This hosts a send-and-receive API only connected to the scanner instances and no other systems. This also helps to map the AI to instance requests to avoid confusion and mixups. This is the second largest system deployed, having more to do with GPU-intensive tasks.

Connections and Protocols: The connections are made via API requests as each is accessed by either 1 or 2 of the systems at a given time. They are assigned IPs according to that particular system. The connections are made over HTTPS to avoid MITM or sniffing attacks if any intrusion into the system occurs. The passwords used must be hashed with custom hashes to avoid any breaches.

Security Groups: The security policy on this system is to be specific, and that is: This allows the AI instance IP to only connect to the scanner instance over port 443 and disallows any other Ingress access. Allowing the Scanner instance IP to send and receive data to and from the AI instance, the database, and the Interface instance over port 443 will also disallow any other Ingress access. Allowing the Interface IP to connect to the Scanner instance over port 443 and to the system operator over 22 and disallow any other Ingress access. Here, we can see all of the operations happening over HTTPS and the AI and Scanner instances with no SSH internet access to ensure they are not accessed. This also relates to the database access as the scanner can only access it. This effectively reduces the risk factors. While some do exist, this reduces the overall risk to the system.

3.7. Benefits of Traditional systems

The following are the benefits of this system over the traditional decentralization approach:

- **Segmented and Controlled Access:** You establish a controlled environment by breaking the system into separate components (interface, scanner, artificial intelligence, and database) and limiting their interactions. Doing this reduces the possibility of a single point of failure, and the potential harm from a security breach is also constrained.
- **Concentrated Resource Distribution:** Every element is customized for its distinct function. For example, the AI system uses much GPU power for AI tasks, whereas the Scanner system is optimized for CPU and storage to manage multiple docker instances. Doing this ensures that resources are used effectively where they are most needed and are not wasted.
- **Enhanced Security Measures:** Strong security measures include using custom hash algorithms for password protection and HTTPS for connections. These prevent data interception and unauthorized access, including Man-in-the-Middle (MITM) attacks.
- **Limited Network Access:** You can lessen the surface area for possible attacks by restricting network access (e.g., AI and Scanner instances without SSH internet access). This is especially crucial in decentralized systems since the risk of exposure can frequently rise with the number of nodes.
- **Selective Inter-Component Communication:** The system's architecture makes certain interaction paths (such as an AI instance connecting exclusively to the Scanner instance over port 443) possible. This further improves security by lowering complexity and guaranteeing that each component only communicates with its required counterparts.
- **Scalability:** This configuration is more easily scalable due to its modular design, which assigns unique roles to each instance. For instance, more Scanner instances can be added without affecting other components if more scanning capacity is required.
- **Customizable Security Policies:** A more robust and specialized security framework is made possible by the ability to set distinct security policies for each component (e.g., different access permissions for the AI instance, Scanner instance, and interface).
- **Database protection:** Restricting access to the database to the scanner alone eliminates the possibility of data breaches and unauthorized data manipulation by preventing direct external access.
- **Overall Risk Reduction:** By separating functions, restricting access, and utilizing secure communication protocols, this strategy considerably lowers the overall risk, even though no system can be completely risk-free.

4. Simulation Analysis

The simulation focused on testing the collaboration between the API and the AI within a decentralized system, emphasizing integration rather than the intricate details of AI development. Specifically, the assessment involved running scans on seven vulnerable machines to evaluate the system's collective efficiency, responsiveness, and accuracy in operating in two distinct environments. It's crucial to note that the paper does not delve into the active development or detailed testing of the AI's

accuracy; instead, the primary objective is to assess the responsiveness of both the API and AI models within the decentralized system. The range of machines tested encompassed various Linux and Windows operating systems, maintaining consistent internet conditions. The scanning parameters included scanning all top 10,000 ports with the scanner set to operate using Nmap in four TCP scanning modes. The emphasis lies on comprehensively testing the system's overall responsiveness, encompassing the collaboration between the API and AI, the time taken for their collective operation, and their overall efficiency, with a dedicated focus on integration within the decentralized framework. Using the simulations, the following data was collected:

- **Name:** The Name of the vulnerable machine scanned.
- **Total Time:** The total time taken for the analysis to complete.
- **GPT AI Time:** The individual time the AI takes to complete its analysis.
- **API Time:** The total time the API takes to run the network scan.
- **Accuracy:** The accuracy of the final result with bare regular Nmap.

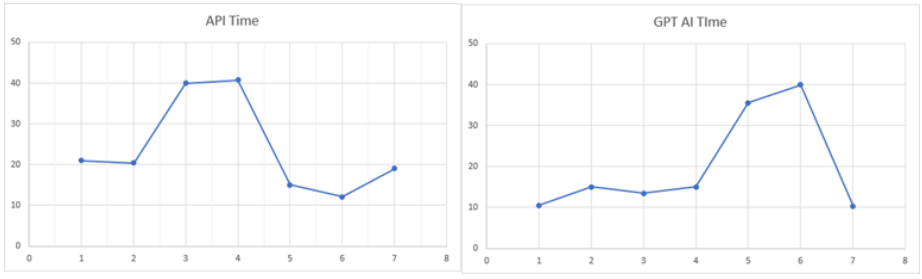
The reason for the absence of accuracy calculation is due to:

- The system does not focus on AI development.
- The accuracy does not consider the false negative or false positive values for the details provided.
- The percentage represents the detection of services, ports, and severity.

Table 1: Scan Results

Name	Time complete	GPT AI Time	API Time
CM1	31.57	10.57	21
CM3	35.41	15	20.41
Tomcat	53.45	13.45	40
Recon	55.77	15	40.77
Metasploitable	50.6	35.6	15
Cyborg	52.07	40	12.07
WordPress	29.36	10.36	19

Even without an accuracy rate for each of the tested systems, it appears from this data that the system can evaluate the vulnerability of various systems with a high degree of accuracy. The system is also fairly quick, taking between 29.36 and 55.77 seconds to evaluate the vulnerability for WordPress and Recon, respectively. We are concentrating mainly on the speed and operational proof that this system works and can be implemented in a real-world scenario. The advantages of such a system are endless when considering the significant upgrades in the likes of AI. There seems to be some difference across the many systems evaluated regarding the breakdown of GPT AI and API components (Table 1). For instance, the GPT AI component analyzed the CM1 system vulnerability in 10.57 seconds, while the API component analyzed the Tomcat system vulnerability in the same time (40 seconds). However, in every instance, the system's two parts could finish their analysis in an acceptable amount of time, and the results were dependably accurate. When seen graphically, we get the following: In Both Figure 7a and 7b, it appears that the



(a) API Time Taken (b) AI Time Taken

Figure 7: Combined Time Taken Figures

API time and AI time refer to the time it takes the system to finish performing the vulnerability analysis and enumeration using two different techniques, one of which uses the GPT AI and the other a third-party application programming interface (API). For instance, the vulnerability analysis and enumeration were finished in 31.57 seconds for the CM1 simulation. The API took

21 seconds to finish the analysis, compared to the GPT AI's 10.57 seconds. Like the GPT AI, the Tomcat simulation required 53.45 seconds to complete the analysis, whereas the API needed 40 seconds. The data indicates that the GPT AI completed the analysis more quickly than the API in general, with the AI time being shorter than the API time for most simulations. It is crucial to remember that the time difference between the two approaches varies throughout simulations, with some simulations exhibiting a greater time difference than others. The data indicates that the GPT-3-based AI-based vulnerability analysis and enumeration method may produce results more quickly and effectively than conventional API-based techniques. We have also conducted tests implementing the system on a smaller scale and having an operational test conducted. In this simulation, 11 sets of analyses are done, each with 10 minutes of data logging the CPU usage, RAM usage, and time taken. The time ranges from zero to eleven, each consisting of 10 minutes. The total CPU allocated is 10 Cores, and the total RAM is 15GB.

In this simulation analysis, we examine the performance metrics of a Garbage Collection (GC) system on various sets. For each set, the data shows the amount of time (in minutes), the percentage of CPU time used, and the percentage of memory used. We observe a steady decline in CPU and memory usage over time, which speaks to the effectiveness of the GC system. When processing the first set (set 1), the CPU utilization is 54.5%, and the memory utilization is 30.54%; these metrics progressively decrease as more sets are processed. This pattern shows how the GC system can recover.

Table 2: Instance Performance Benchmark

Instance	Time (in min)	CPU (in per cent)	Memory (in per cent)
Set 1	0	54.5	30.53
Set 2	1	46	34.32
Set 3	2	46	33.95
Set 4	3	43	33.94
Set 5	4	42	33.87
Set 6	5	44	33.79
Set 7	6	42.5	33.78
Set 8	7	40	33.78
Set 9	8	40	33.84
Set 10	9	39.5	33.91
Set 11	10	38	33.83

And maximize resources over time, leading to increased effectiveness. Significantly, by set 11, the CPU utilization drops to 38%, demonstrating how well the system manages computational resources. There is also a decrease in memory utilization; by set 11, it reaches 33.83%. Overall, the simulation analysis highlights how well the GC system works to optimize CPU and memory resources gradually, improving system performance (Table 2).

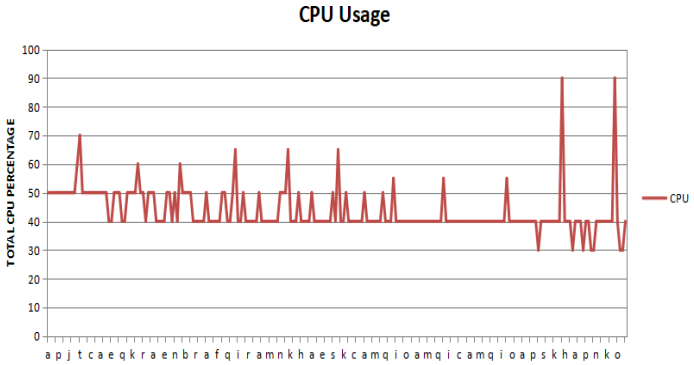


Figure 8: CPU Performance

In Figures 8 and 9, we can see the main stability and operational issues. While looking at 8, we can see that the information is based on the different instances deployed and mapped according to each instance's runtime CPU maximums over the running of the time duration sets. As mentioned before in the methodology, the MAX LOAD, CPU, and RAM values can be predefined, and we had defined it as 90% for the CPU and 75% for the RAM, whereas the average LOAD could be in the 70-80 range. Whenever the load reached the 90% mark, the GC system rebooted and immediately got the CPU usage down by 40% and down to 30% while restarted. 9, the CPU drops near the set 7 and 8 plots. This graph is a direct plot of the above data in Table 2. This is an average of all the instances.

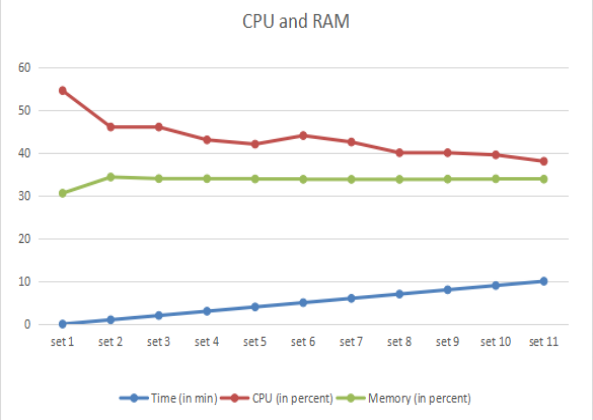


Figure 9: Total benchmark

Running at the time and the combined CPU and RAM values. This graph shows that time steadily increases while CPU and RAM usage are maintained constantly.

Table 3: Instance Performance Benchmark

Cycles	CPU Load (in %)	RAM Load (in %)	Overall load (in %)
1	150.9359	2.3791	76.75
2	151.5219	2.3398	76.0815
3	151.2218	1.8743	76.548
4	151.2218	1.8743	76.548
5	149.7572	2.9633	76.3602
6	150.3486	2.2722	76.3104
7	149.3672	1.3922	75.5387
8	149.3995	2.5693	75.9844
9	151.1149	2.9014	77.0082
10	149.903	2.8511	76.3802
11	149.7473	1.1309	75.4391
12	149.2987	1.7184	75.5086
13	152.01457	1.8859	76.9502
14	151.2429	2.855	77.049
15	151.9381	2.1415	77.0398
16	149.1872	1.5047	75.346
17	149.7772	2.8333	76.3052
18	149.4406	2.1868	75.8137
19	150.3554	4.0103	77.7828
20	150.9062	2.7437	76.8249

In Table 3, we deliberately prepared a docker image designed to take all the resources, and we ran five such instances simultaneously while recording the output. The output we desire is that after the max threshold we have set and all the conditions have been met, the instances will be stopped from being executed. The threshold we used is the combined load must not exceed 60% because we need to see how much of an effect it can have on our deployment. With this implementation, we can see that CPU usage has risen to 150 per cent, and that's a massive amount, considering that RAM is at 2%. Even though the odds are so different, we can see that the load is on the higher end of the spectrum, and this is good as we need a clear view of the load, which shows how even one misuse of resources can be detected accurately. This record was done over 20 cycles, with each cycle having five such instances running and collecting the CPU and RAM usage information.

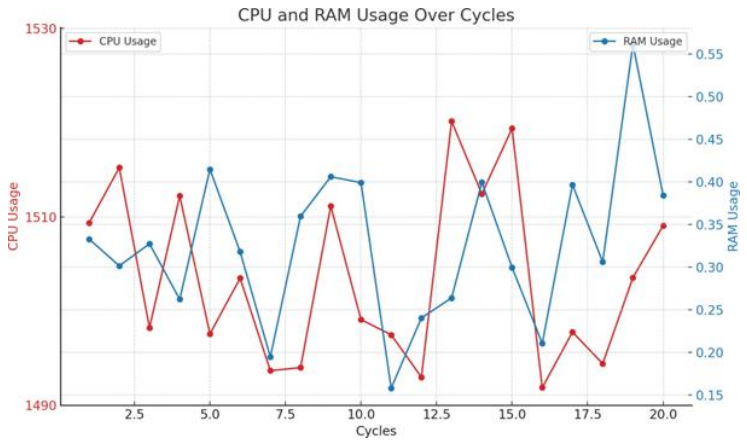
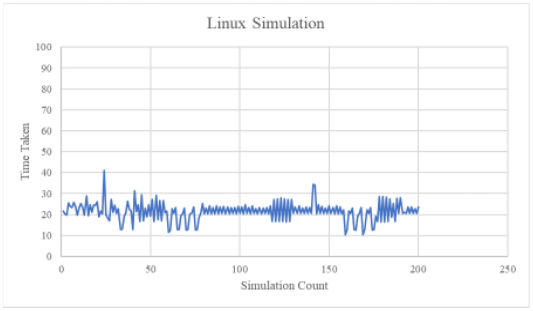
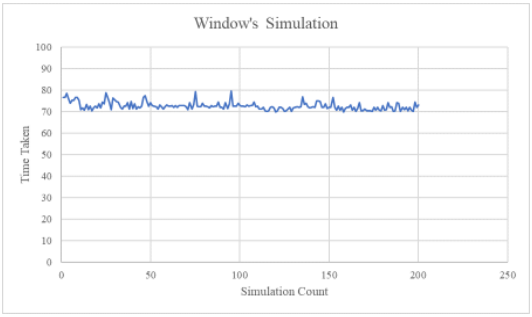


Figure 10: CPU And RAM usage over the instances

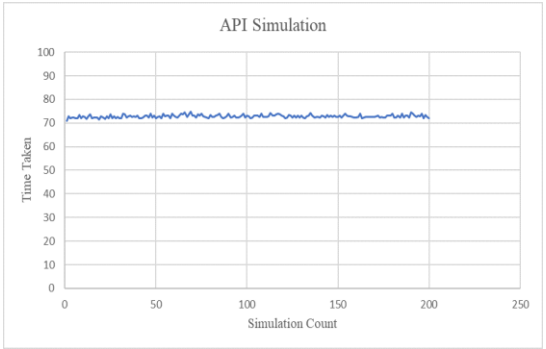
From Figure 10, We can see how Docker interprets the usage scores, which for the CPU is in the range of 1490-1530% and 0.15-0.55%, but the Load formulation calculates and converts this to an overall value and individual Load value which is in the 0-100% range. This graph shows how much stress the instances are putting on the system while running, and the GC functionality helps rectify this issue. From our previous research, the paper proved that using Dockerization for the scanner API improved the system's overall stability. The data from our previous research had 600 simulations run under stress and at runtime. The average output is listed in 4. Based on a validated performance assessment of scanning simulations, Windows showed stability and consistency, with scan times consistently falling between 70 and 80 seconds over 200 scans. Unlike Windows, Linux displayed faster but wildly inconsistent scan times, ranging from 10 to 43 seconds, which resulted in instability and unreliability. Running on a Linux server, the Nmap API found a balance between scan durations of 70-75 seconds, providing slightly faster speeds than Windows and more stability than Linux. This demonstrated the API's effectiveness in balancing speed and stability (Figure 11a,b,c).



(a) Linux Stability

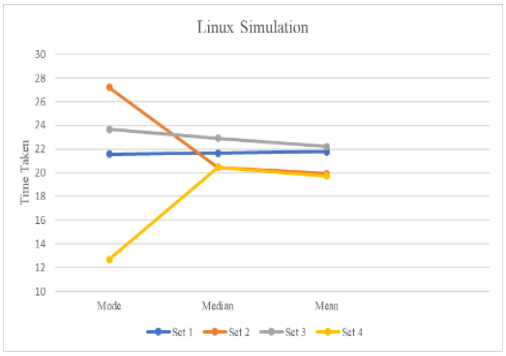


(b) Windows Stability

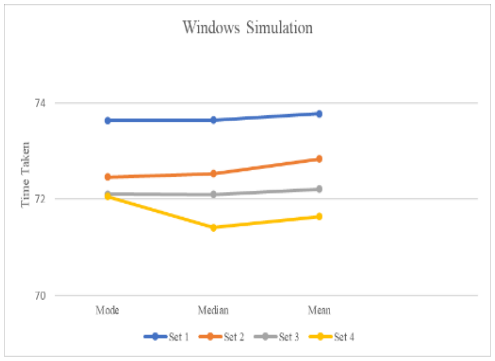


(c) API Stability

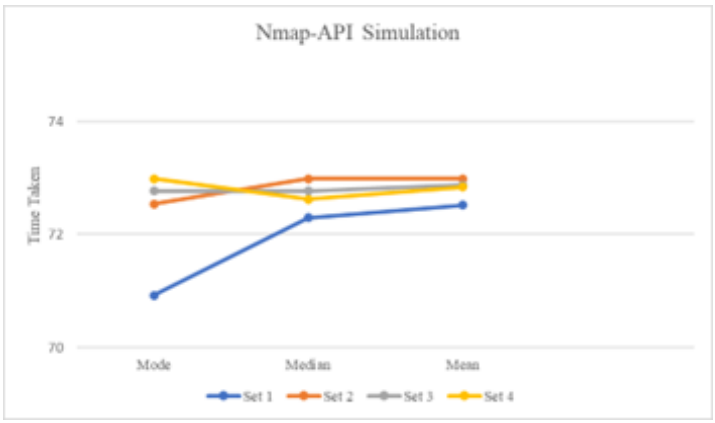
Figure 11: Combined Stability Figures



(a) Linux Performance



(b) Windows Performance



(c) API Performance

Figure 12: Combined Performance Metrics

Table 4: Dockerized Instance Stability

Set's	Window	Linux	Nmap API
SET 1	Mode: 73.64 Median: 73.645 Mean: 73.7804	Mode: 21.5754 Median: 21.6579 Mean: 21.8701	Mode: 70.921 Median: 72.2962 Mean:72.5158
SET 2	Mode: 72.46 Median: 72.2962 Mean: 72.5158	Mode: 27.2018 Median: 20.4745 Mean: 19.9406	Mode: 72.539 Median:72.9846 Mean: 72.9885
SET 3	Mode: 72.11 Median: 72.1 Mean: 72.2150	Mode: 23.6684 Median: 22.9101 Mean: 22.20778	Mode: 72.7704 Median:72.7692 Mean: 72.8737
SET 4	Mode: 72.06 Median: 71.41 Mean: 71.6405	Mode: 12.682 Median: 20.4624 Mean: 19.7366	Mode: 72.9880 Median: 72.6254 Mean: 72.8361

Nmap API showed consistent timing, with most scans finishing in 72–74 seconds and the fastest being 70 seconds, according to an analysis of 50 scan sets from four computers. There was very little difference between scans—between 0.5 and 0.7 seconds, at most (Figure 12a,b,c). This consistency illustrates how effectively the API functions even when managing several user requests simultaneously. The Nmap API is faster than Windows but slower than Linux, finding a compromise between stability and speed. Its use of a Docker instance is responsible for its stable performance. The Nmap API is a better option in real-world situations due to its balance of speed and stability—it is faster than Windows and more reliable than Linux (Table 4). The simulation analysis of the Garbage Collection (GC) system's performance across different sets reveals a consistent and encouraging trend. The observed decrease in CPU and memory usage over time highlights the system's effectiveness in

resource reclamation. Notably, the first set shows the system's capacity to gradually optimize resources, with a CPU utilization of 54.5% and a memory utilization of 30.54%. By setting 11, the CPU utilization even lowers to 38%, demonstrating the system's efficient allocation of computational resources. Further insights are revealed by the graphical representations in Figures 8 and 9, which show operational and stability problems. The automated reboot feature of the system effectively reduces high loads to 30% CPU usage by activating at a predetermined threshold of 90% CPU usage. The second graph also displays the system's overall performance, showing consistent time increases with constant CPU and RAM utilization. This thorough examination validates the GC system's resilience in boosting effectiveness and preserving stability across the simulated sets.

5. Conclusion

AI-powered Decentralized Vulnerability Assessment with Port Scanning is a powerful choice for companies looking to strengthen their security. The decentralized design of this system increases its resistance to potential cyber threats and guarantees reliable performance. It lessens the need for stringent security checks by enabling safe communication between network nodes and doing away with the requirement for a centralized governing body, freeing up resources for other innovations and projects. One important feature of this system is that it operates quickly and reliably, which contrasts sharply with the lengthy timelines involved in traditional vulnerability assessments. Advanced AI algorithms, especially those built on GPT-3 technology, can be integrated to quickly and accurately identify vulnerabilities. This flexibility is essential for reducing cyber threats and quickly fixing security flaws, which lowers the likelihood of cyberattacks and data breaches. This system is always changing, incorporating state-of-the-art AI research areas like reinforcement learning for adaptive security strategies, machine learning for anomaly detection, and natural language processing for threat intelligence analysis. These AI-driven techniques offer a flexible and adaptable security framework by improving the precision and effectiveness of vulnerability assessments. But just as important as appreciating the system's advantages is acknowledging its shortcomings. Even with GPT-3's and similar AI technologies' sophistication, careful human oversight is still required to confirm the validity of vulnerability detection results. This is necessary to guarantee the reliability of the results and reduce the possibility of false positives or negatives.

The AI-supported Decentralized Vulnerability Assessment with Port Scanning is a strong option for businesses looking to improve their security protocols. Its decentralized architecture strengthens its defences against possible intrusions while ensuring efficient operation. This method frees up resources for other important projects by simplifying security procedures and permitting safe, decentralized communication. The decentralized system performs consistently well thanks to the ongoing integration of AI research and development. The continued emphasis on system optimization and AI developments highlights the dedication to providing enterprises with dependable, advanced security solutions. Subsequent enhancements will investigate novel AI domains and optimize the system, demonstrating a resolute commitment to progressing security technologies.

Acknowledgment: The authors express their sincere gratitude to their co-authors for their valuable contributions, expertise, and commitment, which significantly enhanced the quality of this work. The authors also thank their friends for their continuous encouragement and support throughout the research journey. Institutional support from Jain (Deemed-to-be University), Bengaluru; Presidency College, Kempapura, Bengaluru; Sharda University; and Northumbria University, London, United Kingdom, is gratefully acknowledged.

Data Availability Statement: The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Funding Statement: This research received no external funding. The study was conducted, and the manuscript was prepared without financial assistance.

Conflicts of Interest Statement: The authors declare no conflicts of interest related to this work. This study is an original contribution by the authors, and all referenced materials have been properly cited.

Ethics and Consent Statement: The research was conducted using established ethical standards. Informed consent was obtained from all participants, and confidentiality was maintained to ensure participant privacy.

References

1. M. Barrere, R. Badonnel, and O. Festor, "Vulnerability Assessment in Autonomic Networks and Services: A Survey," IEEE Communications Surveys Tutorials, vol. 16, no. 2, pp. 988–1004, 2014.

2. K. Kertysova, “Artificial intelligence and disinformation: How AI changes the way disinformation is produced, disseminated, and can be countered,” Security and Human Rights, vol. 29, no. 1–4, pp. 55–81, 2018.
3. S. A. Harp, J. Gohde, T. Haigh, and M. S. Boddy, “Automated Vulnerability Analysis Using AI Planning,” in AAAI Spring Symposium: AI Technologies for Homeland Security, Stanford, California, United States of America, pp. 46–53, 2005.
4. M. Malkawi, T. Özyer, and R. Alhajj, “Automation of active reconnaissance phase: an automated API-based port and vulnerability scanner,” in Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Hague, Netherlands, pp. 622–629, 2021.
5. K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, United States of America, pp. 311–318, 2002.
6. D. Frey, J. Royan, R. Piegay, A. M. Kermarrec, E. Anceaume, and F. Le Fessant, “Solipsis: A decentralized architecture for virtual environments,” in 1st International Workshop on Massively Multiuser Virtual Environments, Reno, NV, United States of America, 2008.
7. R. Abhishek Reddy, G. Chiranjeevi, Eshwar Guptha, Z. Mohammed Zaid, “AI Based Enumeration and Exploit Suggester,” International Journal of Emerging Technologies and Innovative Research, vol. 9, no. 6, pp. 208–212, 2022.
8. L. Floridi and M. Chiriatti, “GPT-3: Its nature, scope, limits, and consequences,” Minds and Machines, vol. 30, no.11, pp. 681–694, 2020.
9. R. Sun, J. Yang, and M. Yousefzadeh, “Improving Language Generation with Sentence Coherence Objective,” 2020, Pre-Print.
10. E. J. Hu et al., “Lora: Low-rank adaptation of large language models,” arXiv pre-print, arXiv:2106.09685, 2021, Press.
11. J. Sklansky, “Computational discovery for nonlinear classifiers,” Proceedings of the 1992 IEEE International Conference on Systems, Man, and Cybernetics, Chicago, IL, USA, vol. 2, no.10, pp. 1308–1313, 1992.
12. S. Singh and N. Singh, “GPT-3.5 vs. GPT-4, Unveiling OpenAI’s Latest Breakthrough in Language Models,” 2023, Pre-Print.
13. G. Chiranjeevi, R. Abhishek Reddy, and R. Shyam, CVE LLM Dataset [Computer software], 2023. Available: https://github.com/morpheuslord/CVE-llm. [Accessed by 10/03/2024]
14. G. Chiranjeevi, R. Abhishek Reddy, and R. Shyam, GPT vuln analyzer [Computer software], 2023. Available: https://github.com/morpheuslord/GPT-Vuln-analyzer. [Accessed by 10/03/2024]
15. G. A. Cavaliere, R. Alfalasi, G. N. Jasani, G. R. Ciottone, and B. J. Lawner, “Terrorist Attacks Against Healthcare Facilities: A Review,” Health Security, vol. 19, no. 5, pp. 546–550, 2021.