ROYAL MILITARY ACADEMY

163rd SSMW Promotion

Colonel René Bauduin

Academic Year 2025 – 2026

3rd Bachelor

Teaching Web Security Through Vulnerabilities: A Comparative Study of Vulnerable Web Applications

By Adjudant COC Robrecht DE ROO

Bachelor's Thesis at CISS Department
Presented for the obtaining of the bachelor's degree
in Social and Military Sciences
Under supervision by Lieutenant-Colonel Thibault DEBATTY
Brussels, 2026

Abstract

The increasing reliance on web-based information systems within military and governmental contexts has increased the importance of cybersecurity education. Vulnerable Web Applications (VWAs) are widely used as practical training tools to teach web application security concepts in a controlled and legal environment. However, despite their widespread availability, clear comparative guidance on their respective features, educational value, and deployment approaches remains limited. Therefore, this thesis addresses the following research question: "How do different publicly available Vulnerable Web Applications compare to each other in terms of key features such as vulnerabilities, educational effectiveness, and deployment approaches?"

To answer this question, a literature-based comparative analysis was conducted. Publicly available VWAs were selected from the OWASP Vulnerable Web Applications Directory using clearly defined selection criteria. Each selected VWA was analysed using a comparison framework focusing on three dimensions: vulnerability coverage, educational effectiveness, and deployment and maintenance characteristics.

The results show that VWAs vary significantly in design philosophy and intended use. While some applications prioritise structured, lesson-driven learning, others emphasise realism and reproducibility. Therefore, no single VWA provides an optimal solution for each and every context.

In conclusion, the choice of a VWA should be based on the learner's experience level, educational objectives, and deployment constraints. This study provides a structured overview that can assist educators in selecting the most appropriate tools for cybersecurity training.

Contents

1 Introduction

1.1 Context and motivation:

The importance of information systems used in the context of military operations and infrastructure is becoming greater than ever before: the use of hybrid warfare through methods such as cyberwarfare is becoming ever more prevalent in today's conflicts. As such, it has become a major point of attention for large military organisations like NATO, who are interested in developing ways to defend themselves or use it in their favour [1].

Today's battlefields are already managed using complex systems which, despite them being built to withstand attacks, can still contain vulnerabilities. In the battlefields of tomorrow, the danger of these all-important systems being hacked becomes even greater as a hacked system will lead to even greater casualties. Off the battlefield, the risks are also high: if a malicious actor manages to hack his way into one of Belgian Defence's many important systems, like for example our Human Resources or Material Resources departments, the results could be catastrophic in nature. The only way to counteract this danger is to be able to defend our information systems against any foreign meddling. This can only be achieved by reinforcing our information systems against any possible vulnerabilities, and if any are detected, by fixing them as fast as possible. Thus, it is also an imperative that the educational institutions that are linked to Belgian Defence are able to instruct their information systems personnel as best as possible, as they represent the first and last line of defence against electronics-based threats. It is for this reason that a comprehensive study of the various Vulnerable Web Applications on the market today is of immense value for Belgian Defence, as this will allow educators to better choose the tools that they will use to teach the subject of programming vulnerabilities [1].

1.2 Research Question*:*

There are many Vulnerable Web Applications available online, but the differences between them are not clearly indicated, thus the research question for this thesis is as follows: "How do different publicly available Vulnerable Web Applications compare to each other in terms of key features such as vulnerabilities, educational effectiveness and deployment approaches?"

1.3 Scope and limitations:

This research will take the form of a literature-based comparative analysis wherein I will evaluate publicly available Vulnerable Web Applications (which will be referred to as VWA from here on out). The main reference I will use as foundation of this study is the OWASP (Open Worldwide Application Security Project) Vulnerable Web Applications Directory (VWAD) as it already categorises VWAs based on multiple criteria. I will accompany this with peer-reviewed publications, project documentation made by the

different VWA developers as well as any supplementary OWASP resources that are relevant to the study. I will base this study exclusively on online sources as the subject of coding and web applications, as well as the vulnerabilities within them, keep evolving every day and these sources will therefore be the most up to date ones. There is a limitation as this is a literature study: I will not be deploying any VWAs myself for the purpose of adding information to said study.

This thesis begins by introducing background information by giving important definitions, as well as some context and history about VWA and web applications in general. I will then detail the methodology I have chosen in order to correctly compare the chosen VWA. Next, I will catalogue the chosen VWA and provide all the relevant information. Following that will be a comparative analysis, detailing the actual differences between VWAs. Afterwards, I will interpret the findings and conclude this thesis.

2 Background

2.1 Definition of VWA:

There is no officially designated definition for Vulnerable Web Applications. It is a piece of software that runs in a web browser that should not be mistaken with a website due to a key difference: whereas a website provides static content that users download from a server to then simply "browse", usually in the form of a HTML page or an image file; a web application allows users to interact with dynamic data in various ways [2]. Most commonly, this is done through media like forms, authentication systems, or interactions with a database. VWAs deliberately contain security flaws or weaknesses that can be exploited by "attackers" in a controlled environment, so they can learn to identify, fix or exploit said vulnerabilities [2].

2.2 History and Evolution of Web Vulnerabilities:

To give some context to the importance of VWA on today's internet, it is important to go over a brief history of the evolution of online vulnerabilities and of the internet itself.

As described by Islam and Xiangdong [3, p. 3], the internet started in the 1960s as a project by the Department of Defense's Advanced Research Projects Agency of the United States of America entitled ARPANET. This computer network's main objective was facilitating communication between academic institutions. As the project was not sufficiently advanced yet, security measures were hardware-based, not software-based [3, p. 3].

During the 1970s, software-based vulnerabilities grew in visibility as the very first programs targeting vulnerabilities (and the applications used to combat them) were written. At the same time, advancements were made in the domain of cryptography, a domain which is, to this day, extremely important for online security measures [3, p. 4].

In the 1980s, computers were a very popular item on the consumer's market. These came with the possibility of having an internet connection, as well as the dangers associated with that. As many more citizens were getting access to the internet, vulnerabilities were being discovered faster, and they also had bigger consequences. As a result, the first companies specialising in computer security appeared. Their primitive antivirus programs used "signature-based detection methods" [3, p. 4] which is only effective against viruses that exploit known vulnerabilities [3, pp. 4-5].

The internet of the 1990s was a lot more advanced than its previous iteration 10 years earlier. It was characterised by an increased commercialisation, the appearance of "web-based services" [3, p. 5], and the invention of the World Wide Web by British computer scientist Timothy Berners-Lee. Due to the newfound interactivity of the web, vulnerabilities were seen as a more critical aspect that needed to be addressed as their impact could be much greater and could lead to far more damage than before [3, pp. 5-6].

With the presence of more vulnerabilities came more countermeasures, with one of the most significant events being the appearance of the Open Web Application Security Project (OWASP) in 2001. The organisation helped developers to make their code more resistant to threats and vulnerabilities by teaching them how to identify these vulnerabilities [3, pp. 6-7].

The arrival of OWASP was just in time for the internet of the 2000s, where many different types of malware, as well as the increasing amount of Distributed Denial of Service (DDoS) attacks caused the internet in this decade to be quite unstable. Cybersecurity had to make technological leaps forward to stay ahead of hackers. The 2000s saw advancements in different techniques and protocols like Transport Layer Security (TLS) or in antivirus software, which implemented new detection methods beyond the simple signature-based detection [3, pp. 7-8].

Today, the use of Artificial Intelligence (AI) tools is becoming increasingly commonplace as AI is a lot more efficient and thorough in scanning through massive amount of code [3]. Nevertheless, these advanced technologies should still be overseen by competent developers who will still need to be trained in the identification of vulnerabilities. This underlines the importance of the development and use of VWA.

2.3 Common web vulnerabilities:

There are a great number of different types of vulnerabilities present on the internet, therefore providing an exhaustive list here would not be useful. Instead, the 10 most important web application vulnerabilities are detailed in the "OWASP Top 10" which was last updated in 2025. What follows is a brief description of each vulnerability [4].

The most dangerous vulnerability according to OWASP (which has maintained its top spot from the 2021 update) is Broken Access Control, this vulnerability permits a user to perform actions for which they do not have the necessary permissions, such as requesting information that should only be accessible by an authenticated user or an administrator [4].

The second most dangerous vulnerability is the Security Misconfiguration. As the name implies, this means that a web application's security measures are incorrectly configured. A few examples are the presence of unnecessary features, much backward compatibility leading to vulnerabilities in the system or the lack of adequate hardening [4].

The next vulnerability is Software Supply Chain Failures, where the use of outdated third-party software, or third-party software component that has been tampered with by individuals or groups seeking to compromise systems that use that specific software component [4].

The fourth vulnerability on the list are Cryptographic Failures. The use of outdated cryptographic algorithms or simply the lack of encryption (i.e.: storage of data in

Plaintext) of sensitive data means that if hackers get access to a database behind a web application, they can take the sensitive data with very little resistance [4].

The fifth vulnerability is Code Injection, this category includes Cross-Site Scripting (XSS), SQL injection, and Command Injection. In this vulnerability, a malicious actor interacts with the web application by entering information into a field that requires it. For this vulnerability to work, the field must not filter the input correctly (or at all). By inputting code that was specifically written to make use of this type of vulnerability, the hacker can force the application or database to send sensitive information to them [4].

The sixth vulnerability, Insecure Design, is focused on architectural flaws in the program's design. "This includes flaws in the business logic of an application, e.g. the lack of defining unwanted or unexpected state changes inside an application" [4].

The seventh is Authentication Failures, where a malicious actor manipulates the web application into believing the incorrect login information entered is actually valid, thereby authenticating the hacker and allowing them to access data as if he were logged-in as a user or administrator [4].

The eighth on the list is "Software or Data Integrity Failures" [4], described as follows: "Software and data integrity failures relate to code and infrastructure that does not protect against invalid or untrusted code or data being treated as trusted and valid. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline without consuming and providing software integrity checks can introduce the potential for unauthorized access, insecure or malicious code, or system compromise" [4].

The penultimate entry on this list is "Security Logging & Alerting Failures". While not a vulnerability in the sense that a hacker could use it to directly access a system, if the system has insufficient or no logging of errors or interactions, then the malicious entity can freely test the application's defences without ever triggering an alert in the owner's system, greatly lowering the level of security [4].

The final of the 10 most dangerous vulnerabilities is a new category added in the 2025 edition of the top 10: Mishandling of Exceptional Conditions. There are three types of mishandlings as defined by OWASP: "the application doesn't prevent an unusual situation from happening, it doesn't identify the situation as it is happening, and/or it responds poorly or not at all to the situation afterwards" [4]. These can lead to many different security vulnerabilities that take the form of "logic bugs, overflows, race conditions, fraudulent transactions, or issues with memory, state, resource, timing, authentication, and authorization." [4] which are then exploited by attackers [4].

To sum up, the OWASP Top 10 provides a comprehensive overview of the most dangerous web application security risks that are currently most widespread. While not an exhaustive list, it shows the vulnerabilities that are most relevant to modern web applications. Because many VWA are explicitly designed to showcase vulnerabilities from

one or more OWASP Top 10 categories, this list will serve as a reference throughout this thesis when analysing and comparing the vulnerability coverage of different VWA.

2.4 VWAD and the educational role of VWA:

The Vulnerable Web Application Directory (VWAD) by OWASP is a comprehensive, centralised, curated and actively maintained registry of publicly available VWA (and vulnerable mobile applications) that are available for anyone to learn the basics of web application security. As the single most authoritative list of VWA currently available, it is from this database that the selection of VWA that will be reviewed will be made [5]. VWA provide a safe environment for aspiring security experts, from novice to more advanced user, to legally practise hacking a web application. These can be deployed in an offline environment, containerised or online [6].

2.5 Additional important information

To talk about VWA and their deployment methods, it is necessary to introduce several supporting technologies and applications and how they are used in the deployment of modern web applications and of VWA.

One particularly important technique is Operating-System-level virtualisation. There are multiple applications that make use of this technique like Docker, Podman or Guix. Docker will be used as example for this thesis. It is described as "an open platform for developing, shipping, and running applications" [7]. Written in the Go programming language, Docker allows a developer to package and execute his software in a "container" which separates it from the underlying infrastructure of the host system [7]. It achieves this by using OS-level virtualisation instead of a virtual machine's hardware virtualisation [8]. Whereas a virtual machine emulates a completely separate computer system (going as far as emulating the hardware), a container uses the host computer's operating system (more specifically, the same kernel) as a base to run pieces of software in. This makes it a lot more lightweight than a full virtual machine, leading to easier and less resource-intensive deployment, thereby making it more suitable for applications like VWA [7] [8].

While containerised applications are relatively easy to manage, this quickly becomes a lot more complex when a developer needs to manage a large number of different containers. To remedy this problem, it is possible to apply a container orchestration framework [9]. As defined by The Kubernetes Authors in the official application documentation: "Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services that facilitate both declarative configuration and automation" [9]. This framework is often used in conjunction with an application like Docker to automate a lot of processes like deployment, scaling, and management of the different containerised applications [10].

When containerisation is insufficient or inappropriate for the deployment of an application or the needs of a particular developer, a virtual machine may be employed. There exist many applications for this purpose but one frequently recommended by VWA developers is Vagrant by HashiCorp, a subsidiary of IBM [11]. While not strictly a virtual machine itself, it acts as a management layer for other "providers" [11] like VirtualBox VMware, Hyper-V. The "providers" handle the actual hardware virtualisation [11]. The use of actual virtual machines can ensure that more obscure operating systems can be used without compatibility problems that can occur in containerisation [12]. As stated by HashiCorp in the official documentation: "Docker will not provide the same production parity as a tool like Vagrant. Vagrant will allow you to run a Windows development environment on Mac or Linux, as well." [12].

Another important platform to define is Node.js, as it is used extensively in intentionally insecure applications. The OpenJS Foundation defines Node.js as an "open source and cross-platform JavaScript runtime environment" [13] which is primarily used to develop server-side applications [13]. It is built to handle input/output operations such as network requests, file access requests or database queries asynchronously, which means that these requests do not block the program, allowing it to continue processing other tasks simultaneously [13]. Because of this efficient and scalable design, it is often employed in authentication services and real-time communication systems, where theoretically millions of users could access a web application simultaneously [14]. Because of its extensive use online, many VWA use Node.js as the underlying runtime so as to be as realistic as possible when compared to actual web applications [13].

Finally, ZAP and Burp Suite, two widely used web application security testing tools. The first of the two, created by Checkmarx, is free and open source. It is known as a "manipulator-in-the-middle proxy" [15] as it intercepts and inspects data that is sent from the browser to the web application, and vice-versa. It can then read, modify, and forward those data-packets to its intended destination [15]. It is adapted for use by users with differing skill levels, from beginners to experts, and it can be installed on all the major operating systems, as well as on Docker [15]. The second one, Burp Suite by PortSwigger, is a proprietary (closed source), commercial piece of software. It provides the same functionality as ZAP (i.e. intercepting data-packets between a browser and a web application), but it also provides advanced tools such as automated vulnerability scanning, request sequencing analysis or session handling testing [16]. Although there is a free version, it is extremely limited, therefore users are encouraged to invest in the 'professional' edition of the software [16]. These security testing tools allow developers, cybersecurity experts and researchers to evaluate the effectiveness of the vulnerabilities that are present in VWA. For this reason, many VWA require the use of one of these applications.

3 Methodology

3.1 Selection criteria for VWAs:

This thesis will focus exclusively on VWA as defined by OWASP: web applications that are intentionally designed to contain security vulnerabilities for the purpose of legal education, training, and security research [6].

To ensure methodological consistency, the VWA analysed in this study are selected according to the following criteria.

First, all selected VWA must be listed in the OWASP Vulnerable Web Application Directory. Using the VWAD as a baseline ensures community recognition as the VWA can be sorted by popularity. This popularity is indicated by the amount of "stars" a VWA has received from the community [5]. A minimum threshold of 100 stars must be attributed to a particular VWA for it to be selected. This threshold is to ensure that it is worthwhile to take this VWA into account for a comparative analysis.

Second, there needs to be intentional vulnerability and explicit educational purpose. Only applications that are explicitly documented as intentionally vulnerable and designed for educational or training use are considered. This excludes public web applications that are accidentally vulnerable, as well as other software not intended to be exploited by users for educational purpose. The intentional nature of the vulnerabilities ensures that all testing activities are legal and ethically justified.

Third, the application must primarily demonstrate web application–level vulnerabilities, such as authentication flaws, injection vulnerabilities, access-control weaknesses, or misconfigurations. Applications that focus exclusively on infrastructure, cloud account misconfiguration, or general system hardening without a web application component are excluded. The OWASP Top 10 will be used as initial reference list to measure how many types of vulnerabilities a VWA can present.

Fourth, the selected VWA must provide sufficient functional and architectural complexity to allow meaningful analysis of vulnerabilities, educational effectiveness, and deployment considerations. This will be based on factors such as the number of features present or the diversity of different types of vulnerabilities. Conversely, extremely minimalistic VWA or VWA whose only purpose is to demonstrate a single vulnerability are excluded, as they cannot be compared properly to other VWA.

Fifth, to ensure relevance to modern web technologies and security practices, selected applications must show evidence of active or recent maintenance, such as updates, documentation revisions, or ongoing community engagement. Abandoned or obsolete applications are excluded, even if they are listed in the VWAD. VWA that have not been updated in the last 6 months will be considered as obsolete. The threshold of 6 months was chosen to make sure the project is modern enough while also taking into account that these educational projects are not updated very regularly [5].

The final selection criterion is that the chosen VWA must collectively represent a diverse range of technologies, architectures, and deployment approaches. This diversity will support comparative analysis and prevents the study from being biased toward a single programming language, framework, or learning style.

3.2 Comparison framework:

To be able to compare the VWA, the previous criteria are bundled into a list of following key criteria, which will be analysed for each one in order to paint a clear picture of the similarities and differences between all the chosen VWA.

Firstly, 'vulnerabilities': what type of vulnerabilities, and what vulnerabilities specifically are modelled in each VWA and how they compare to each other. Are there only the most prevalent and modern vulnerabilities or does the VWA also model less well-known vulnerabilities, are these vulnerabilities quite simple or more complex?

Then, the 'educational effectiveness' as measured by the user-friendliness of the VWA: are there tutorials, difficulty settings, how in-depth and complete is the project documentation, how realistic are the vulnerabilities (and are they presented in a way that is in accordance with what could be found on an actual web application online, or are they more "gamified" to ease learning?). Does the project allow for a Capture The Flag (CTF), this is defined as follows by Rohit Jha from Meusec.com [17]: "[it] is a type of challenge given to cybersecurity professionals and students within a time period in which they have to apply their skills, knowledge over technologies to hack the victim machine". If the project has specific learning objectives, how do they compare to the objectives of other VWA? Is the user interface clear, and is it fixed, or can it be customised by the user? Finally, do they only target beginners, or do they offer challenges for more advanced users of experts in the field?

Last but not least, the 'deployment and maintenance'. The type of setup and the accompanying advantages and disadvantages of said deployment type (system architecture, online or offline, platform, sandboxing features, etc.), as well as the level of maintenance of the VWA. As vulnerabilities are discovered and patched away, a scarcely maintained VWA could quickly become outdated, therefore, an actively developed VWA if paramount. How active the community is and how fast they respond to any queries will also be analysed when possible.

3.3 Use of Artificial Intelligence

Large Language Models (LLM) were used at the start to help gather sources for this thesis. The writing of the thesis was done by me with only occasional help from an LLM (Microsoft Copilot) when something needed to be reformulated more academically, and to check for eventual grammatical or spelling errors. The author takes full responsibility for the content of this work.

4 Overview of selected VWAs

4.1 OWASP Juice Shop:

OWASP's Juice Shop is the first VWA on the list as it is the most popular VWA in use right now according to the VWAD [5]. Released in its first form in October 2014, it is the self-proclaimed "most modern and sophisticated insecure web application" [18]. The VWA is written in Node.js, Express and Angular and the developers describe it as an application that can be used for training, or as a "guinea pig-application" [18] that security experts can use to test their tools on [18].

4.1.1 Vulnerabilities:

Juice Shop contains all vulnerabilities found in the OWASP Top 10. This includes the numbers one and two: "Broken Access Control" and "Security Misconfiguration" [4]. Other highly ranked vulnerabilities are also present but have received a slightly different name, such as separating Code Injection into "Injection", "XSS", and "XXE" ("XML external entity" (XXE) attack, where the targeted vulnerability resides in an XML input). Some vulnerabilities listed on their website are not found in the OWASP Top 10, such as "Broken Anti Automation", which are vulnerabilities that target checks (like CAPTCHA) to make sure a user is actually human and not an automated script or a bot [18].

4.1.2 Educational effectiveness:

Juice Shop is a textbook example of a gamified VWA. It provides 111 "challenges" of varying difficulty, with a user's score being uploaded to a scoreboard. Their difficulty is indicated with a score based on the amount of stars a challenge has, the score being between 1 and 3. The 1-star challenges are mostly aimed at beginners, and 13 of the 111 challenges are designated as "tutorial challenges" to help a novice find out how to use and abuse the VWA. On their own website [18], the challenges are sorted into "categories", which indicate the type of vulnerability the student will be facing; as well as so-called "tags" which can help guide the student towards the right path for solving the challenge. [18]

The challenges proposed by Juice Shop can be solved in multiple different ways, which is positive for the development of the skills of users who already have a basic understanding of the different vulnerabilities. This does, however, increase the difficulty for novices who have no prior knowledge on how to approach a certain vulnerability, if they do not make use of the built-in tutorials [18]. The application tracks which challenges have been completed, notifies the user, and saves this progress even when the application is restarted and wiped clean. [18]

The Juice Shop platform also supports Capture The Flag (CTF) through the use of an additional tool by the same developers entitled MultiJuicer, allowing users to run multiple instances of Juice Shop. This can be used in a classroom-environment to foster a learning environment where classmates can help each other and compete against each other to advance their skills [19].

The application's appearance is extensively customisable to adapt its appearance and functionality to a user's or organisation's needs through the modification of a YAML configuration file [20].

4.1.3 Deployment and Maintenance:

The Juice Shop project completely free and open source, and is published under the MIT license. It is very flexible in the ways it can be installed. The VWA can be installed locally on a user's computer (by using a prepacked format or cloning from source), it can be installed in a containerised environment through the use of Docker or Vagrant, and it can also be used online as the developers also host an online version of their VWA [18]. Lastly, it comes preinstalled on OWASP's SamuraiWTF, a Linux virtual machine destined to be used for application security training [21].

The Juice Shop is very actively maintained, as the latest commit on GitHub, as of writing this thesis, was on April 14th 2026, and the latest release (v19.2.1) released on March 7th 2026 [22].

4.2 WebGoat:

The next VWA on this list is OWASP's WebGoat. It is one of the most long-standing VWA available, as its earliest GitHub release dates back to 2016 [23]. The application is mainly written in JavaScript and as stated in the OWASP Developer Guide, "provides an environment where a Java-Based web application can be safely attacked without traversing a network or upsetting a website owner" [2].

4.2.1 Vulnerabilities:

According to OWASP, WebGoat presents lessons for almost all of its Top 10 vulnerabilities, alongside a few that are not in the Top 10 [24]. The provided lists of vulnerabilities are those of the OWASP Top 10 of 2017 [24] and 2021 [23], nevertheless it is likely that the VWA has been updated to reflect the most current edition of the Top 10 (2025), as the project is actively being maintained, whereas the OWASP's site describing the VWA may not be [24].

4.2.2 Educational effectiveness:

Contrary to Juice Shop's approach, WebGoat is built on a framework of guided lessons, where each vulnerability has its own subsection. The application informs the user of what is expected of them to succeed in the current assignment and allows them to proceed to the next one when the user succeeds [23]. Each vulnerability is broken down into multiple assignments to make sure the user can tackle progressively more advanced hurdles [23].

4.2.3 Deployment and Maintenance:

Like the previous entry on the list, WebGoat is free and open source, and published under the GNU GPL-2.0 license [23]. The application cannot be executed online but can only be

installed locally on a user's computer through the installation of a prepackaged Java Archive file or through cloning the source from GitHub. The application can also be run through the use of Docker, which is the preferred method of running the application if the user already has ZAP or Burp installed on their system [23].

WebGoat is actively maintained, with the last commit on GitHub being dated to March 19th 2026 [23].

4.3 DVWA:

The third VWA on this list is one that is not developed by OWASP. It is a web application running on PHP with a MariaDB backend [25].

4.3.1 Vulnerabilities:

While the DVWA repository does not cite which vulnerabilities are present, the 'dvwa-vulnerabilities' project by James Cao [26] catalogues the vulnerabilities found inside the VWA. According to this project, the following vulnerabilities are present in DVWA: authorisation bypassing, weak defences against brute forcing, command injection, weak cryptography, content security policy bypassing, cross-site request forgery, unfiltered file upload, JavaScript attacks, open http redirecting, SQL injection, weak session IDs and finally cross-site scripting (XSS) [26].

4.3.2 Educational effectiveness:

Each of the previously mentioned vulnerabilities have difficulty levels adapted to different levels of user [26]. These are classed in four categories: low, medium, high and impossible difficulty, which facilitates adapting the difficulty to the user's skill level [26].

4.3.3 Deployment and Maintenance:

DVWA is completely free and open source, published under the GPL-3.0 license, and it can be installed on Windows and Linux. For computers running a Debian-based distribution, there is an automated installation script available on GitHub which was not made by but endorsed by the developer of DVWA. For other Linux distros, the VWA can also be installed through the prebuilt Docker image. For Windows, the developer recommends installing it with "XAMPP Apache + MariaDB + PHP + Perl" [25] (XAMPP) as this is supposedly the easiest way to install it [25].

DVWA is very actively maintained, as the three last commits on GitHub are from the 17th, 18th and 19th of March 2026.

4.4 Mutillidae II:

Mutillidae II is a VWA written in PHP. It was developed by Jeremy 'webpwnized' Druin based on the Mutillidae Classic by Adrian Crenshaw [27].

4.4.1 Vulnerabilities:

Mutillidae II contains more than 40 deliberately introduced vulnerabilities. These vulnerabilities map to multiple editions of the OWASP Top 10, specifically the 2007, 2010, 2013, and 2017 versions, and include all major classes of web application weaknesses represented in those lists [28].

4.4.2 Educational effectiveness:

When it comes to educational effectiveness, not much information is found in online literature about how the VWA functions on an educational level. The developer claims that "Mutillidae provides an accessible web hacking environment suitable for labs, security enthusiasts, classrooms, CTFs, and vulnerability assessment tool targets. It has been widely used in graduate security courses, corporate web security training, and as an assessment target for vulnerability assessment software" [29], but no extensive list of features was found after research [29].

4.4.3 Deployment and Maintenance:

Mutillidae II is free and open source and is officially distributed under the GNU General Public License version 3.0. It can be installed on both Linux and Windows systems using standard LAMP (Linux + Apache + MySQL + PHP), WAMP (Windows + Apache + MySQL + PHP/Perl/Python), or XAMPP stacks, and it is also available as a prebuilt Docker image for easier deployment in classroom environments. Additionally, it comes preinstalled on 'Rapid7 Metasploitable 2' [28], a virtual machine built specifically to be vulnerable [30]; on OWASP SamuraiWTF, in conjunction with OWASP's Juice Shop [21], [28]; and on OWASP's Broken Web Applications (BWA) Project, which is a (slightly outdated) virtual machine based on VMware that also comes with VWA preinstalled [31].

The project is actively maintained as the GitHub repository shows regular commits, with the used versions of PHP and Apache being updated as of late April 2026 [29].

4.5 WrongSecrets:

WrongSecrets is another OWASP developed VWA. This one is focused specifically on 'secrets management', as it is designed to demonstrate common mistakes in storage and handling of sensitive data inside software that are used in identification measures like passwords, API or 2FA keys, and session tokens, to name a few [32]. It is written primarily in Java, using the Spring Boot framework. Spring Boot is an open source framework that can be used for many different objectives. In this context it is used to assist in programming web applications by providing a robust framework for developers to build on [33].

4.5.1 Vulnerabilities:

As mentioned previously, WrongSecrets focuses on insecure secrets management practises. Examples of these insecure practices include hard-coded secrets in source code,

improperly protected configuration files, secrets exposed through environment variables, insecure use of Kubernetes secrets, secrets stored in Docker images, and misconfigurations in cloud infrastructure templates [34]. Some challenges also involve secrets managed through external systems such as HashiCorp Vault or cloud-native secret stores, where the misconfiguration lies in access control or integration rather than storage alone [34].

4.5.2 Educational effectiveness:

WrongSecrets offers its users 65 different challenges [32]. These challenges each have a focus such as AI, Binary, Cryptography, Docker, Git, Password Manager, Secrets or Web3, to name a few. This allows the users to select which type of challenge they want to try to expand their knowledge in a particular domain [32]. The challenges are ranked through a difficulty level between one and five, one being the easiest (i.e. basic code inspection) and five being the very hardest challenges. This means users can follow a logical order in which to try the challenges [32]. Finally, the environment in which the challenge is based upon, with most of the challenges running on Docker [32]. It is possible to host a CTF on WrongSecrets through the use of a fork of Multi Juicer entitled 'WrongSecrets CTF Party' [35]. It is also possible to use the WrongSecrets platform to test the effectiveness of secret detection tools [34].

4.5.3 Deployment and Maintenance:

WrongSecrets is free and open source and is published under the GNU APGL-3.0 license [34]. The application can be deployed locally using Docker or executed in more complex environments such as Kubernetes clusters or public cloud platforms, depending on which challenges are required. The developers also host an online demo of their VWA on their own website, allowing interested users to try it out before committing to installing the application on their system [32].

The official GitHub repository shows active development. As of early May 2026, the latest commits include updates to existing challenges, dependency upgrades, and documentation improvements [34].

4.6 Security Shepherd:

OWASP Security Shepherd is a web and mobile application security training platform developed as an OWASP project. It is written entirely in Java. According to the OWASP Foundation, it is intended to foster and improve application security awareness across users with varying skill levels by providing an environment in which web and mobile security risks can be explored through lessons and challenges [36].

4.6.1 Vulnerabilities:

Like most of the OWASP's own VWA, Security Shepherd uses the OWASP Top 10 as its reference for implementing different vulnerabilities. It implements all of the Top 10 vulnerabilities, as well as some vulnerabilities from the OWASP Mobile Top 10 [36], [37].

4.6.2 Educational effectiveness:

According to the official OWASP Security Shepherd page, the VWA boasts a wide topic coverage with over 70 'levels' covering the "entire spectrum of Web and mobile application security" [36]. Each of these security concepts is explained in layman's terms, with difficulty increasing slowly, allowing beginners to learn at their own pace [36].

The VWA also claims to use very realistic examples, with the emulated security risks being based on real vulnerabilities that have been detected on public websites [36].

It is perfectly suited for use in a classroom environment, as it supports a number of very useful features such as a user management system, allowing admins to exert control over the users as well as check their progress. It has an integrated scoreboard, an integrated feedback system on each level and very detailed and descriptive logging without it being too verbose [36].

4.6.3 Deployment and Maintenance:

Security Shepherd is also open source and free, and it is published under the GPL-3.0 license [38].

The VWA is built to be deployed on Docker or through the use of Security Shepherd's own virtual machine.

4.7 OWASP VulnerableApp:

The OWASP VulnerableApp is a VWA developed by SasanLabs. It is slightly different from all the previous VWA as it is less focused on teaching web application vulnerabilities and more on being a modular platform for benchmarking security tools like ZAP or Burp Suite [39].

4.7.1 Vulnerabilities:

The VWA is capable of simulating a number of different vulnerabilities, some of OWASP Top 10, but some more obscure [40]. The application supports well known vulnerabilities such as Code Injection and Cryptography Failures (from the OWASP Top 10), but also less commonly seen ones like SSRF (Server-side Request Forgery), IDOR (Insecure Direct Object Reference) or Clickjacking [40].

The official project documentation emphasises that vulnerabilities are designed to be deterministic and reproducible. This ensures consistent behaviour across scans, making the application particularly suited for benchmarking and comparative evaluation of security tools [40].

4.7.2 Educational effectiveness:

Contrary to most of the previous VWA, VulnerableApp is not made with beginners in mind. The application is intended to be used by engineers and researchers for the purpose

of improving security testing tools [39]. The platform's modular design allows them to focus on specific vulnerability types or scanner capabilities, such as detection accuracy, false positives, and regression behaviour across versions [39].

While this architecture may also support self-directed learning and manual exploration, OWASP documentation explicitly states that VulnerableApp primarily functions as a tool validation and experimentation platform, rather than as a gamified or lesson-driven training application [39].

4.7.3 Deployment and Maintenance:

OWASP VulnerableApp is free and open source, and it is distributed under the Apache 2.0 license. The application can be executed either as a standalone Java Archive (.jar) file or deployed using Docker, allowing users to run the application locally with minimal configuration. It is also possible to build the application as a Spring Boot application but this is a legacy feature [40].

The project is actively maintained. The GitHub repository shows very frequent commits and regular releases. The most recent commit was on the 10th of May 2026, while the most recent release (end of March 2026) is the version 2.0.0 of the application, introducing numerous enhancements to the platform [40].

4.8 Broken Crystals:

The penultimate VWA on the list is Broken Crystals, a vulnerable application developed by NeuraLegion. It is a very modern VWA, with its development having started as recently as 2023 [41]. Contrary to most of the other VWA on the list, it is not written in Java or PHP, but in TypeScript [41]. It combines a React-based frontend with a Node.js backend while also integrating User Interface elements from Swagger and GraphiQL [41], [42]. It's use case is similar to the previous VWA, as Broken Crystals is meant to function as a benchmarking tool for security applications [41].

4.8.1 Vulnerabilities:

The official GitHub repository outlines the very broad range of security vulnerabilities. Documented vulnerabilities include authentication and session management flaws, such as broken JSON Web Token (JWT) handling, weak or missing signature validation, and misconfigured token claims [41]. In addition, the application contains classic injection vulnerabilities, including SQL injection, LDAP injection, XPath injection, command injection, and server-side template injection, as well as client-side vulnerabilities such as reflected, stored, and DOM-based cross-site scripting (XSS) [41].

4.8.2 Educational effectiveness:

Just like the previous VWA, Broken Crystals does not provide structured lessons or challenges. It is meant to serve as a realistic testing ground where users can manually find and exploit vulnerabilities or deploy automated security testing tools. While it is

possible to use the VWA in a manner similar to other VWA on this list, the documentation focuses exclusively on testing realism and a wide coverage of vulnerabilities, and not on educational guidance [41].

4.8.3 Deployment and Maintenance:

Broken Crystals is free and open source and distributed under the MIT license [41]. The GitHub repository provides Docker configurations that allow users to deploy the application locally, including the backend API, frontend client, and supporting services such as a database [41].

As is to be expected from a modern VWA, it is actively maintained, as the most recent commits are from early May 2026 [41].

4.9 Kubernetes Goat:

Kubernetes Goat (KG) is not a VWA in the same sense as all the previous VWA: it is a vulnerable Kubernetes cluster environment. Unlike the previous VWA that focus on application-layer vulnerabilities, KG is specifically focused on vulnerabilities that can arise inside the Kubernetes framework [43].

4.9.1 Vulnerabilities:

KG has based its selection of vulnerabilities on the OWASP Kubernetes Top 10, mirroring the standard OWASP Top 10. All 10 of the most dangerous Kubernetes vulnerabilities are represented inside KG, with the top 3 vulnerabilities being Insecure Workload Configurations, Supply Chain Vulnerabilities and Overly Permissive RBAC (Role-Based Access Control) [43].

4.9.2 Educational effectiveness:

KG is a scenario-based VWA. It presents users with 22 different scenarios, and each scenario introduces a specific vulnerable configuration or an insecure workload that users can interact with [43].

The documentation found online is extensive, with detailed explanations on the architecture of the VWA, tutorials on Kubernetes, step-by-step walkthroughs of each scenario where it is explained how these insecure situations arise and how to identify and fix them [43].

4.9.3 Deployment and Maintenance:

KG is free and open source, and it is distributed under the MIT License [44].

The project requires access to a functioning Kubernetes cluster and relies on common Kubernetes tools such as ‘kubectl’ and ‘Helm’ for deployment, alongside Docker. Installation is performed by cloning the source from the GitHub repository and running

the setup scripts provided in the repository, which deploy the vulnerable resources into the user's target cluster [43].

5 Comparison

5.1 Summary table:

VWA	Built on	Teaching style	Strengths	Weaknesses	Adapted to/for
OWASP Juice Shop	JavaScript – Angular - Node.js - Express	Gamified challenges and CTFs	Complete OWASP Top 10 coverage, extensive customisation and strong community support	Not enough guidance for complete beginners	Self-paced learning, classrooms, CTFs (with MultiJuicer)
OWASP WebGoat	JavaScript - Java	Guided lessons	Clear learning trajectory with progressively increasing difficulty	Less realistic due to less free-form design, limits creativity	Beginners, academic courses and structured education
DVWA	PHP - MariaDB	Challenges with different difficulty levels	Simple to setup, adjustable difficulty	Very minimal guidance, very little documentation, outdated design	Higher level beginners to high level users
Mutillidae II	PHP (LAMP/WAMP /XAMPP)	Unknown – Not found in any documentation	Ease of setup, widely deployable and realistic attack surface	Lacking documentation and structure	Security enthusiasts in Labs, classrooms, CTFs
WrongSecrets	Java – Spring Boot	65 Challenges made to guide users	Focus on secrets management with support for CTFs	Narrower range of vulnerabilities than other VWA	Users who want to focus on improving their skills in secrets management
OWASP Security Shepherd	Java	70 levels for users to test their skills	Multiplayer is very well integrated,	The setup is notably heavier, and it is less suited for free-form exploration	Classrooms and users seeking to improve through challenges
Kubernetes Goat	Kubernetes	Scenarios	Complete focus on Kubernetes with extensive documentation and realistic configurations	Setup is complex and requires good prior knowledge of Kubernetes	Users who want to focus on learning to manage Kubernetes vulnerabilities

OWASP VulnerableApp	Java – Spring Boot	Deterministic modular framework for security application benchmarking	Reproducible vulnerabilities, modular design, ideal for tool evaluation	Only usable with prior knowledge of cybersecurity, minimal guidance	Security experts, researchers and security tools developers
Broken Crystals	TypeScript (Node.js and React)	Realistic testing environment for security testing	Built on modern technology,	Only usable with prior knowledge of cybersecurity	Security testers and tool benchmarking

5.2 Discussion of results:

In the previous chapter's overview, we see significant differences in design philosophy, pedagogical approach, and intended target audience. While the analysed applications satisfy the selection criteria defined in Section 3.1, the results demonstrate that VWAs are not interchangeable teaching tools, but rather specialised environments designed to address distinct educational or professional needs.

A first observation concerns the relationship between teaching style and learner experience. VWAs such as OWASP WebGoat and OWASP Security Shepherd adopt a highly structured, lesson-driven approach [23], [36]. This means they prioritise clarity, progressively increasing difficulty, and clear learning objectives, making them well adapted to beginners. When compared to applications such as OWASP Juice Shop, DVWA, and Mutillidae II, these offer a more exploratory learning environment [18], [25], [28]. While this approach allows learners more freedom to apply their skills similar to real-world use cases, it also requires more knowledge of cybersecurity, which limits accessibility for newer users.

The comparison also shows a trade-off between realism and educational value. Benchmarking-oriented platforms such as OWASP VulnerableApp and Broken Crystals prioritise reproducibility, realism, and architectural fidelity over education [39], [41]. These environments closely resemble real infrastructures, which makes them more adapted to being used by security researchers and tool developers. However, the lack of structured guidance and learning feedback makes them less suitable for teaching beginners. This is why more guided VWAs may simplify certain elements of real-world applications, thereby reducing realism in favour of clarity and accessibility.

The analysis shows that the vulnerabilities included vary significantly depending on the application's primary objective. General-purpose VWAs like Juice Shop and DVWA aim to cover a wide range of classical web application vulnerabilities, such as the OWASP Top 10 [4], [18], [25]. While specialised platforms such as WrongSecrets or Kubernetes Goat intentionally focus on narrower domains, such as secrets management or container orchestration security. While this limits how many vulnerabilities are included, it enables

greater depth and realism in those specific areas, making such tools particularly suitable for training in specific domains.

Finally, the results indicate that no single VWA can be considered universally optimal. Instead, each platform occupies a distinct position. Selecting an appropriate VWA therefore depends entirely on the intended learner's experience level, learning objectives, and the context in which the training or analysis is performed.

6 Conclusion

The goal of this thesis was to compare publicly available VWAs in terms of vulnerabilities, educational effectiveness, and deployment approaches, with the objective of helping with selecting an appropriate tool for cybersecurity education. Through this literature-based comparative analysis, this study demonstrated that VWAs differ substantially in design philosophy, intended audience, and educational approach.

The results indicate that some VWAs prioritise a guided learning environment that is best suited to beginners, while others emphasise realism, reproducibility, and architectural fidelity, making them more appropriate for advanced learners, researchers, cybersecurity experts or security tool benchmarking. This means no VWA can be considered universally optimal. Instead, the suitability of a VWA depends on the learner's experience level, learning objectives, and technical deployment constraints.

This thesis was limited by its reliance on publicly available documentation and literature, without direct deployment or hands-on experimentation with the analysed applications. As some popular and well maintained VWAs provide little or fragmented documentation, this restricted the depth of analysis in certain cases. Future research could improve this comparison by deploying VWA or performing classroom experiments to assess usability more thoroughly.

7 References

[1] NATO, "HYBRID THREATS AND HYBRID WARFARE - REFERENCE CURRICULUM," Brussels, 2024.

[2] OWASP Foundation, "WebGoat - OWASP Developer Guide," [Online]. Available: https://devguide.owasp.org/en/07-training-education/01-vulnerable-apps/02-webgoat/. [Accessed 2 May 2026].

[3] S. Islam and L. Xiaongdong, "Guardians of the Web: The Evolution and Future of Website," *arXiv,* 2025.

[4] OWASP Foundation, "OWASP Top 10:2025," 2025. [Online]. Available: https://owasp.org/Top10/2025/. [Accessed 14 April 2026].

[5] OWASP Foundation, "OWASP Vulnerable Web Applications Directory," 2026. [Online]. [Accessed 02 April 2026].

[6] OWASP Foundation, "OWASP Developer Guide," [Online]. Available: https://devguide.owasp.org/en/07-training-education/. [Accessed 24 April 2026].

[7] Docker Inc., "What is Docker ¦ Docker Docs," 2026. [Online]. Available: https://docs.docker.com/get-started/docker-overview/. [Accessed 30 April 2026].

[8] A. Pasic, "How Docker Containers Work Under the Hood: Namespaces and Cgroups," 18 April 2025. [Online]. Available: https://www.atlantbh.com/how-docker-containers-work-under-the-hood-namespaces-and-cgroups/. [Accessed 30 April 2026].

[9] The Kubernetes Authors, "Overview ¦ Kubernetes," 2026. [Online]. Available: https://kubernetes.io/docs/concepts/overview/. [Accessed 30 April 2026].

[10] GeeksforGeeks, "Kubernetes Vs Docker," 25 March 2026. [Online]. Available: https://www.geeksforgeeks.org/devops/kubernetes-vs-docker/. [Accessed 30 April 2026].

[11] HashiCorp Developer, "Documentation ¦ Vagrant," 2025. [Online]. Available: https://developer.hashicorp.com/vagrant/docs. [Accessed 30 April 2026].

[12] HashiCorp Developer, "Vagrant vs. Docker ¦ Vagrant," 2025. [Online]. Available: https://developer.hashicorp.com/vagrant/intro/vs/docker. [Accessed 30 April 2026].

[13] OpenJS Foundation, "Introduction to Node.js," 2026. [Online]. Available: https://nodejs.org/learn/getting-started/introduction-to-nodejs. [Accessed 30 April 2026].

[14] L. Orsini, "What You Need To Know About Node.js," ReadWrite, 07 November 2013. [Online]. Available: https://web.archive.org/web/20131111070804/http://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs#awesm=~poWlRvEfa3uMGh. [Accessed 30 April 2026].

[15] Checkmarx, “ZAP,” 2026. [Online]. Available: https://www.zaproxy.org/getting-started/. [Accessed 6 May 2026].

[16] PortSwigger, “Burp Suite Professional,” 2025. [Online]. Available: https://portswigger.net/burp/pro. [Accessed 6 May 2026].

[17] R. Jha, “Introduction to 'Capture The Flags' in CyberSecurity - MeuSec,” 10 June 2020. [Online]. Available: https://web.archive.org/web/20220813175047/https://www.meusec.com/ctf/capture-the-flags-in-cybersecurity/. [Accessed 24 April 2026].

[18] OWASP Foundation, “OWASP Juice Shop,” [Online]. Available: https://juice-shop.github.io/. [Accessed 24 April 2026].

[19] OWASP Foundation, “GitHub - juice-shop/multi-juicer,” 2026. [Online]. Available: https://github.com/juice-shop/multi-juicer. [Accessed 2 May 2026].

[20] OWASP Foundation, “Customisation :: Pwning OWASP Juice Shop,” [Online]. Available: https://pwning.owasp-juice.shop/companion-guide/latest/part4/customization.html. [Accessed 2 May 2026].

[21] OWASP Foundation, “OWASP SamuraiWTF,” 2024. [Online]. Available: https://www.samurai-wtf.org/. [Accessed 4 May 2026].

[22] OWASP Foundation, “Github - juice-shop/juice-shop,” 2026. [Online]. Available: https://github.com/juice-shop/juice-shop. [Accessed 2 May 2026].

[23] OWASP Foundation, “GitHub - WebGoat/WebGoat,” 2026. [Online]. Available: https://github.com/WebGoat/WebGoat. [Accessed 2 May 2026].

[24] OWASP Foundation, “OWASP WebGoat,” 2025. [Online]. Available: https://owasp.org/www-project-webgoat/. [Accessed 2 May 2026].

[25] digininja, “GitHub - digininja/DVWA,” 2026. [Online]. Available: https://github.com/digininja/DVWA. [Accessed 2 May 2026].

[26] J. Cao, “GitHub - jameskaois/dvwa-vulnerabilities,” 2025. [Online]. Available: https://github.com/jameskaois/dvwa-vulnerabilities. [Accessed 3 May 2026].

[27] IronGeek, “Mutillidae,” 2020. [Online]. Available: https://www.irongeek.com/i.php?page=mutillidae/mutillidae-deliberately-vulnerable-php-owasp-top-10. [Accessed 3 May 2026].

[28] OWASP Foundation, “OWASP Mutillidae II,” 2022. [Online]. Available: https://owasp.org/www-project-mutillidae-ii/. [Accessed 3 May 2026].

[29] J. Druin, “GitHub - webpwnized/mutillidae,” 2026. [Online]. Available: https://github.com/webpwnized/mutillidae. [Accessed 3 May 2026].

[30] Rapid7, “GitHub - rapid7/metasploitable3,” 2025. [Online]. Available: https://github.com/rapid7/metasploitable3. [Accessed 4 May 2026].

[31] OWASP Foundation, “OWASP Broken Web Applications Project,” 2016. [Online]. Available: https://sourceforge.net/projects/owaspbwa/. [Accessed 4 May 2026].

[32] OWASP Foundation, “OWASP WrongSecrets,” 2025. [Online]. Available: https://www.wrongsecrets.com/. [Accessed 4 May 2026].

[33] VMware Tanzu, “Spring Boot,” 2026. [Online]. Available: https://spring.io/projects/spring-boot. [Accessed 4 May 2026].

[34] OWASP Foundation, “GitHub - OWASP/wrongsecrets,” 2026. [Online]. Available: https://github.com/OWASP/wrongsecrets. [Accessed 4 May 2026].

[35] OWASP Foundation, “GitHub - OWASP/wrongsecrets-ctf-party,” 2026. [Online]. Available: https://github.com/OWASP/wrongsecrets-ctf-party. [Accessed 4 May 2026].

[36] OWASP Foundation, “OWASP Security Shepherd,” [Online]. Available: https://owasp.org/www-project-security-shepherd/. [Accessed 4 May 2026].

[37] OWASP Foundation, “OWASP Mobile Top 10,” 2025. [Online]. Available: https://owasp.org/www-project-mobile-top-10/. [Accessed 4 May 2026].

[38] OWASP Foundation, “GitHub - OWASP/SecurityShepherd,” 2026. [Online]. Available: https://github.com/OWASP/SecurityShepherd. [Accessed 4 May 2026].

[39] OWASP Foundation, “OWASP VulnerableApp,” 2026. [Online]. Available: https://owasp.org/www-project-vulnerableapp/. [Accessed 10 May 2026].

[40] SasanLabs, “GitHub - SasanLabs/VulnerableApp,” 2026. [Online]. Available: https://github.com/SasanLabs/VulnerableApp. [Accessed 10 May 2026].

[41] NeuraLegion, “GitHub - NeuraLegion/brokencrystals,” 2026. [Online]. Available: https://github.com/NeuraLegion/brokencrystals. [Accessed 10 May 2026].

[42] OWASP Foundation, “Broken Crystals - OWASP VWAD,” 2026. [Online]. Available: https://vwad.owasp.org/app/broken-crystals/. [Accessed 10 May 2026].

[43] M. Akula, “Kubernetes Goat,” 2026. [Online]. Available: https://madhuakula.com/kubernetes-goat/docs. [Accessed 4 May 2026].

[44] M. Akula, “GitHub - madhuakula/kubernetes-goat,” 2026. [Online]. Available: https://github.com/madhuakula/kubernetes-goat. [Accessed 4 May 2026].