ATHENA CTF: A Modular Framework for Instructional Capture-the-Flag Challenges

Zachary Frank

Table of Contents

Table of Figures

Table of Code Snippets

Abstract

Capture The Flag (CTF) challenges are a widely used mechanism for hands-on cybersecurity training, but creating, deploying, and assessing custom challenges remains complex and time-consuming. Existing platforms often rely on static challenge repositories, heavyweight infrastructure, or problems for which solutions have been publicly shared, limiting their suitability for scalable instruction and formal assessment. I present ATHENA CTF, an open source modular framework for the rapid creation, containerization, and deployment of web-based CTF challenges. ATHENA CTF enables challenge authors with modest programming experience to implement fully functional levels in as few as three lines of code, while providing a framework wide standardized execution environment through containerization. The framework supports per-user and per-team solution parametrization, reducing answer sharing and enabling assessment-ready deployments alongside static configurations for demonstrations and collaborative exercises. ATHENA CTF optionally integrates a centralized database to support learner tracking, LMS-based user provisioning, and automated result export. To assist learners without disclosing full solutions, the framework includes optional, context-constrained large language model (LLM) assisted hints generated from limited interaction history and creator-authored solution process. This design balances instructional support with safe-guards against solution leakage and misuse. Together, these features allow instructors and organizations to rapidly prototype, distribute, and assess CTF challenges while maintaining control over deployment, security, and learner interaction.

Keywords: Capture-the-flag, CTF, cybersecurity, education, modular, framework

1 Introduction

In recent years, both the volume and sophistication of cyber incidents have increased substantially [11]. In 2024, Mitre, the organization responsible for the CVE reporting framework, published a record 40,009 vulnerabilities, representing a 38% increase over 2023 [14]. These vulnerabilities have contributed to large-scale data breaches and financial harm, including an estimated $638 million in losses to Canadian citizens in 2024 alone [2; 23]. Together, these trends underscore a persistent challenge: as software systems evolve rapidly, secure development practices often struggle to keep pace [20].

Addressing this gap requires developers to understand not only how vulnerabilities are mitigated, but also how they are identified and exploited. A challenge for this form of education is the pace of technological change, including recent advances in AI-assisted development [6], which necessitate continual updates to security knowledge. Capture-the-flag (CTF) exercises are a widely used approach for delivering hands-on security training, allowing learners to engage directly with realistic vulnerability scenarios [3]. CTFs encourage adversarial thinking and exploratory learning, and can be scaled to accommodate a range of skill levels [15]. However, supporting scalable, up-to-date, and assessment-ready CTF deployments remains a practical challenge.

Large language models (LLMs) have recently demonstrated promise in supporting both self-directed and instructor-assisted learning when provided with appropriate context [10]. Prior work has explored their application in cybersecurity, including exploit detection [16] and educational support [1; 18]. At the same time, the use of LLMs in instructional settings raises concerns around solution leakage, over-scaffolding, and misuse if not carefully constrained.

With this context, I present ATHENA CTF, a containerized framework for authoring, deploying, and assessing web-based CTF challenges. ATHENA CTF uses Docker to provide portable and reproducible challenge environments and offers a modular abstraction for level creation that enables rapid

prototyping and sharing. Challenges can be deployed locally or online and configured for instructional, collaborative, or assessment-oriented use.

The framework optionally integrates a centralized database to support learner tracking, LMS-based user provisioning, and result export, while allowing fully standalone deployments when persistence is not required. To support learner progress without disclosing full solutions, ATHENA CTF includes optional, context-driven LLM-assisted hints. These hints are generated using limited interaction history and creator-authored solution paths, and are intentionally constrained to reduce the risk of solution leakage and misuse.

ATHENA CTF also supports deterministic per-user or per-team parametrization, enabling personalized solutions that reduce trivial answer sharing in formal assessments. Instructors may disable parametrization or LLM-assisted hints to support collaborative activities or demonstrations. To illustrate the framework's expressiveness, we provide a collection of sample challenges spanning introductory to moderate-difficulty web security tasks, including HTTP manipulation and authentication vulnerabilities.

In summary, this work makes three contributions: (1) a lightweight, modular framework for authoring and deploying web-based CTF challenges with minimal setup overhead; (2) assessment-oriented mechanisms, including deterministic per-user parametrization and optional centralized tracking, designed to reduce answer sharing and support formal evaluation; and (3) a constrained approach to LLM-assisted hint generation that balances learner support with safeguards against solution leakage.

2 Related Work

This section reviews related work in security education and compares existing solutions with the design goals and affordances of my framework. I group prior work into three categories: deployment frameworks, CTF learning platforms, and wargame platforms.

2.1 Deployment Frameworks

CTF deployment frameworks generally address two complementary needs: (1) a front-end interface that aggregates challenge links and manages scoring, and (2) a back-end mechanism for deploying interactive levels.

Front-end competition platforms such as CTFd [5] and RootTheBox [13] provide dashboards listing available challenges and award points when users submit correct flags. RootTheBox uses static flags, whereas CTFd supports dynamic flags, but generating them requires paid hosting. In both cases, the delivery of the challenge itself is external; a separate deployment mechanism is needed whenever challenges are not purely static files.

Back-end deployment frameworks such as kCTF [9] and EDURange [22; 24] automate the provisioning of challenges using Kubernetes. To deploy a level, authors must containerize their challenge logic and provide a fully configured Dockerfile. Although powerful, this introduces substantial complexity for instructors or students who lack systems-engineering experience. In contrast, our single-line deployment framework removes the need for custom Dockerfiles, substantially lowering the barrier to creating and publishing web-based challenges.

Other systems, such as Labtainers [12], provide a custom virtual machine (VM) environment encompassing a broad spectrum of security exercises, including networking and systems challenges. Students complete activities within the VM, and instructors later grade exported artifacts. While Labtainers supports a wide range of challenge types, creating new scenarios or deploying lightweight web-based challenges is comparatively cumbersome. Moreover, because Labtainers does not maintain

a database, instructors cannot track live user progress or provide immediate, context-aware feedback; capabilities that our framework is explicitly designed to support.

2.2 Capture the Flag Learning Platforms

Self-directed learning platforms offer persistent CTF-style challenges, typically enhanced with gamification such as points and leaderboards. Unlike ephemeral competitions, user progress is stored over time and challenges remain publicly accessible.

PicoCTF [4] is one of the most widely used platforms, providing challenges ranging from web exploitation to cryptography. Its large library supports skill development, but public availability makes it unsuitable for graded coursework: solutions and walkthroughs are easy to find, and users must create external accounts. Its hint system is static and not tailored to learner context, and flags are not personalized, increasing the likelihood of answer-sharing.

The pwn.college DOJO platform [18; 19] offers an interactive in-browser environment for each challenge, including options for a VSCode interface, a full VM, or a shell. This reduces setup friction but introduces reliance on strong network connectivity and can incur slowdowns from VM overhead. The browser-based environment also limits students working on web-security challenges, who benefit from their personal browsers and extensions. DOJO allows users to host their own instance and create custom challenges, but, similar to Labtainers, authors must fully implement each level, which remains complex for simple web challenges. DOJO also includes LLM-powered context-aware hints, though this approach incurs higher computational cost and a greater risk of prompt injection relative to our lightweight hinting mechanism.

SEED Labs [7] provide structured, self-directed security labs accessible through a custom VM. While effective for teaching Linux-based exploitation, the VM requirement makes even introductory web challenges overly complex for beginners and similarly restricts users to a non-native environment.

2.3 Wargame Platforms

Wargame platforms offer collections of static challenges, often binaries, packet captures, or SSH-based exercises, with minimal scaffolding or instructor feedback. In web security, commonly used platforms include bWAPP [21], DVWA [25], and OWASP Juice Shop [8]. These tools expose learners to standard vulnerabilities, but updates are infrequent and public availability results in widespread write-ups, making them unsuitable for assessing student mastery in formal coursework.

BWapp and DVWA lack centralized data storage, making it difficult to track learner progress or manage multi-user classroom deployments. Although Juice Shop supports multi-instance deployment and progress tracking, extending it to include new challenges requires deep integration with the existing code base. Most instructors, therefore, lack the time or expertise needed to create custom levels, limiting its suitability as a flexible teaching tool. ATHENA CTF addresses this gap since levels can be created, deployed, and monitored with little effort and learning required.

3 Framework Features

In this section, I describe the design of my CTF framework and how it can be leveraged to create modular and distributable levels with context-enabled hints. Additionally, I will explore how the database implementation can be used for assignments and competitions.

3.1 Web-Based Architecture

ATHENA CTF is built as an entirely web-based interaction framework, allowing both challenge creators and participants to deploy and access levels through standard web browsers. Challenges can be distributed either as containerized services or as source repositories deployed directly on bare-metal systems. When containerization is used, challenge images can be shared through standard container registries such as Docker Hub and pulled for local or remote deployment. Alternatively, creators may distribute source repositories, enabling administrators to deploy challenges using provided spawning scripts without requiring container tooling.

The framework is designed to operate across Unix and Windows-based systems, allowing administrators to select deployment workflows that align with their technical comfort and available infrastructure. In contrast to VM-based platforms, ATHENA CTF does not require learners to install additional software or work within preconfigured virtual environments. Once deployed, challenges may be hosted on a local network or exposed to the internet via a reverse proxy, supporting classroom use, workshops, and remote participation.

A web-based architecture was selected to maximize accessibility and realism for web-security tasks. Learners interact with challenges using their native browsers and tooling, enabling direct engagement with HTTP-level vulnerabilities and client-side behaviours without the overhead of managing virtual machines. This design choice also improves compatibility with constrained or low-resource environments, such as high-school or university computer labs, where installing custom software or virtualisation platforms may be impractical.

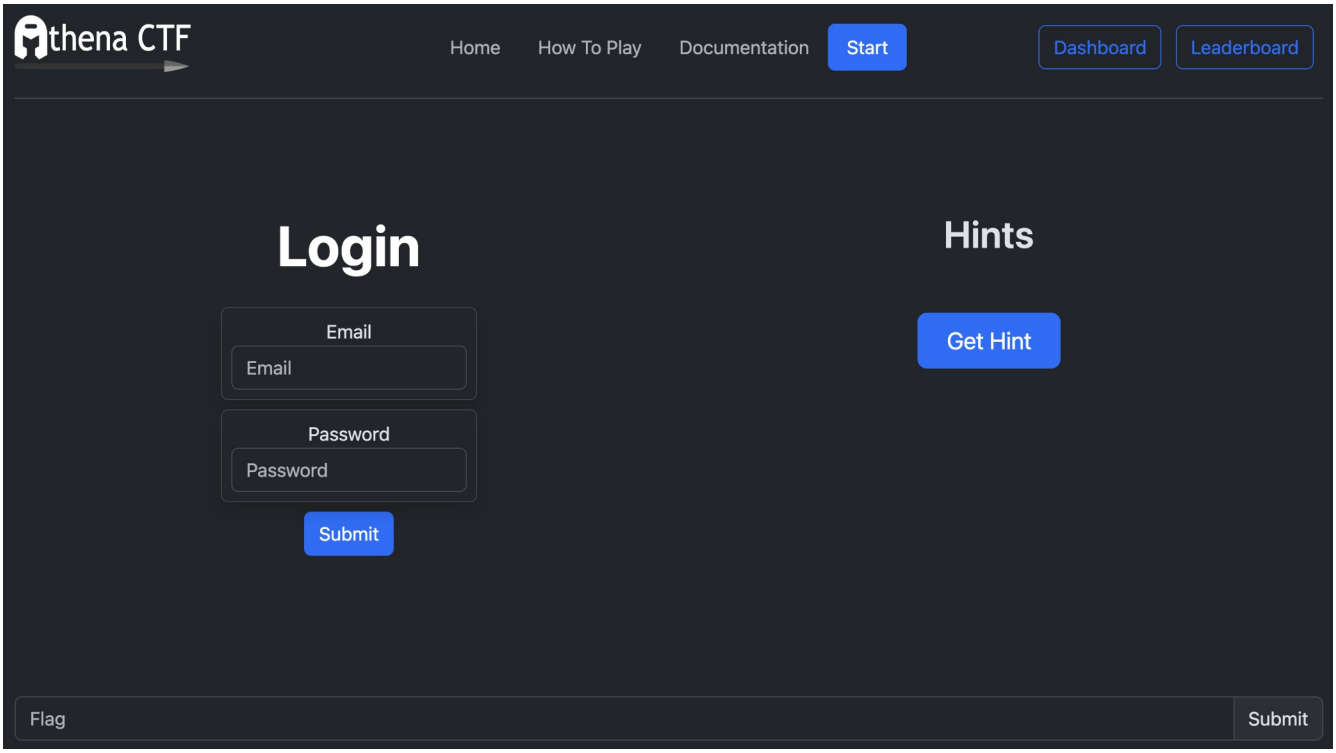

Figure 1: Example ATHENA CTF challenge requiring a participant to bypass a vulnerable login form using native browser interaction. The challenge illustrates the framework's web-based design, which allows learners to engage directly with HTTP-level and client-side vulnerabilities without requiring virtual machines or specialized tooling.

The web-based design further enables flexible configuration of user experiences at deployment time. Administrators may expose different interfaces or hinting mechanisms for the same challenge, such as enabling static hints for collaborative demonstrations or context-driven LLM-assisted hints for individual learners. These configurations can be isolated per user or deployment instance without modifying challenge logic, allowing instructors to adapt challenges to different instructional and assessment contexts.

Figure 1 shows an example challenge in which users must bypass a vulnerable login form using browser-based interaction.

3.2 Centralized Database

ATHENA CTF optionally supports a centralized database to enable learner tracking, assessment workflows, and context-aware assistance. The database is not required for challenge execution and can be disabled entirely for standalone or demonstration deployments. When enabled, it provides a lightweight mechanism for recording user interaction data across challenges while minimizing storage overhead and configuration complexity.

I implement this functionality using a document-oriented NoSQL data model (MongoDB [17] JSON format), as each user's interaction history can be naturally represented as a single, compact record. This design simplifies data access for features such as progress tracking, hint generation, and result export, while avoiding the schema rigidity of relational databases. Administrators enable database support by providing credentials to an existing local or cloud-hosted instance and specifying a target collection; the framework automatically initializes required fields and indexes.

In practice, the storage requirements are minimal. In a demonstration deployment with 11 challenges, the average user record occupied approximately 115 bytes, and the index size was roughly 1 MB across approximately 11,000 users. This footprint allows the framework to scale to large classes or workshops using commodity infrastructure, including free-tier cloud database services.

Database support is required for advanced features such as LLM-assisted hints and automated assessment export, but remains optional to preserve flexibility and ease of deployment. By decoupling challenge execution from persistent storage, ATHENA CTF allows instructors to select configurations that best match their instructional, privacy, and infrastructure constraints.

3.3 LLM-Assisted Hints

ATHENA CTF includes optional, context-aware LLM-assisted hints designed to support learner progress while minimizing the risk of solution disclosure and misuse. Rather than providing an open-ended

conversational agent, the framework generates one-shot plain text hints in response to explicit user requests. This design intentionally constrains interaction with the LLM, reducing exposure to prompt injection attacks, uncontrolled dialogue, and unintended leakage of full solutions.

Hint generation is grounded in two sources of controlled context: the user's recent interaction history and a creator-authored solution path. User interaction data, including prior requests and failed attempts, is retrieved from the centralized database when enabled. The solution path, provided by the challenge author as either a structured write-up or executable script, is stored locally on the host system and loaded into memory at run time. This approach ensures that hints are aligned with the intended solving strategy while preventing the model from inventing alternative or unintended solution paths.

By combining limited historical context with a bounded, creator-defined reference, the LLM is guided to produce targeted guidance related to the user's current progress rather than complete answers. The non-conversational, stateless nature of hint requests further constrains the output space and enables predictable inference costs, making the approach suitable for classroom-scale deployments.

LLM-assisted hints are an optional feature and can be replaced with static hints or disabled entirely through configuration, allowing administrators to select appropriate levels of assistance for demonstrations, collaborative exercises, or graded assessments. The framework supports both commercial LLM APIs and locally hosted models, enabling deployments that align with available infrastructure, cost constraints, and data governance requirements.

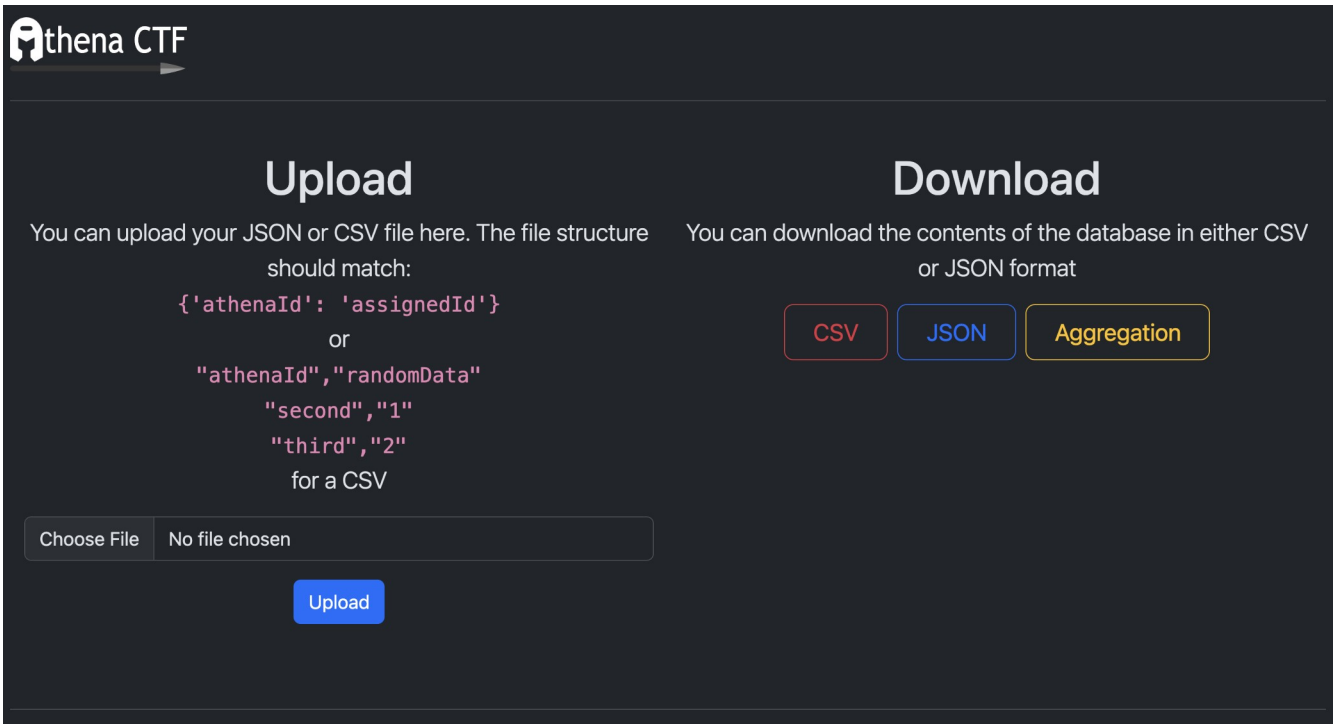

Figure 2: Administration interface used for assessment oriented workflows in ATHENA CTF. Instructors can provision users via LMS-exported identifiers and export challenge results without storing personally identifying information,supporting separation between learner interaction data and institutional records.

3.4 Ease of Assessment

ATHENA CTF provides a dedicated administration interface, deployed as a separate container, that enables instructors to manage users and assess participant performance without modifying challenge logic. This separation allows assessment workflows to be configured independently of challenge deployment, supporting both instructional and evaluative use cases.

By default, users are assigned an anonymous identifier upon first access, which is stored locally as a browser cookie and used to track challenge progress when database support is enabled. For formal assessments, administrators may instead provision users explicitly by uploading a CSV or JSON file exported from a learning management system (LMS). User creation is driven solely by a designated identifier field (e.g., *athenaId*); no additional data from the import file is stored or persisted by the

framework. This design ensures that assessment can be conducted without collecting or retaining personally identifying information.

The same identifier may be assigned to multiple participants to support team-based challenges, allowing progress and submissions to be recorded collectively. Throughout an assessment period, the framework records limited interaction metadata, including challenge completion status, submitted requests, and hint usage. At the conclusion of an assessment, administrators can export the collected data directly from the administration interface. These records can be joined with the original LMS export to support grading and review while maintaining separation between platform data and institutional records.

Figure 2 shows the administration interface used for user provisioning and result export.

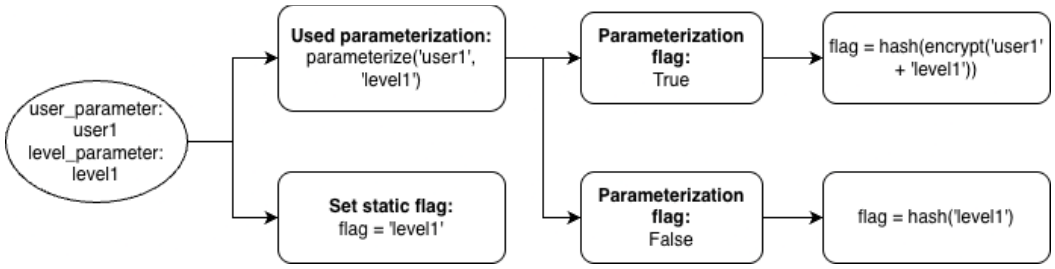

Figure 3: Verification flow for static and parametrized challenge solutions. When parametrization is enabled, user-specific identifiers are combined with a level-specific secret to deterministically generate unique solutions per participant, reducing trivial answer sharing while preserving reproducibility.

3.5 Parameterization

ATHENA CTF includes a parametrization mechanism that enables the generation of deterministic, user-specific solutions on a per-level basis. This design mitigates trivial answer sharing and supports fair assessment by ensuring that each participant must independently solve a challenge, even when working on identical challenge logic.

At level creation time, the framework generates a level-specific secret value. During verification, this secret is combined with a user-specific identifier—derived either from an automatically assigned browser cookie or an administrator-provisioned account —to compute a unique expected solution for that user. As a result, repeated attempts by the same user yield a consistent solution, while solutions shared across users or teams are invalid. This approach preserves reproducibility for individual learners while preventing straightforward collusion.

Parametrization is configurable on a per-deployment and per-level basis. Administrators may disable parametrization to use static solutions for in-class demonstrations or collaborative exercises, or selectively enable it for advanced challenges intended for individual assessment. To support this flexibility without increasing implementation complexity or introducing divergent code paths, the parametrization function is invoked during every flag verification request, regardless of whether

parametrization is enabled. When disabled, the function deterministically produces a static, level-wide solution.

Figure 3 illustrates the verification flow used to compute and validate challenge solutions under both parametrized and static configurations.

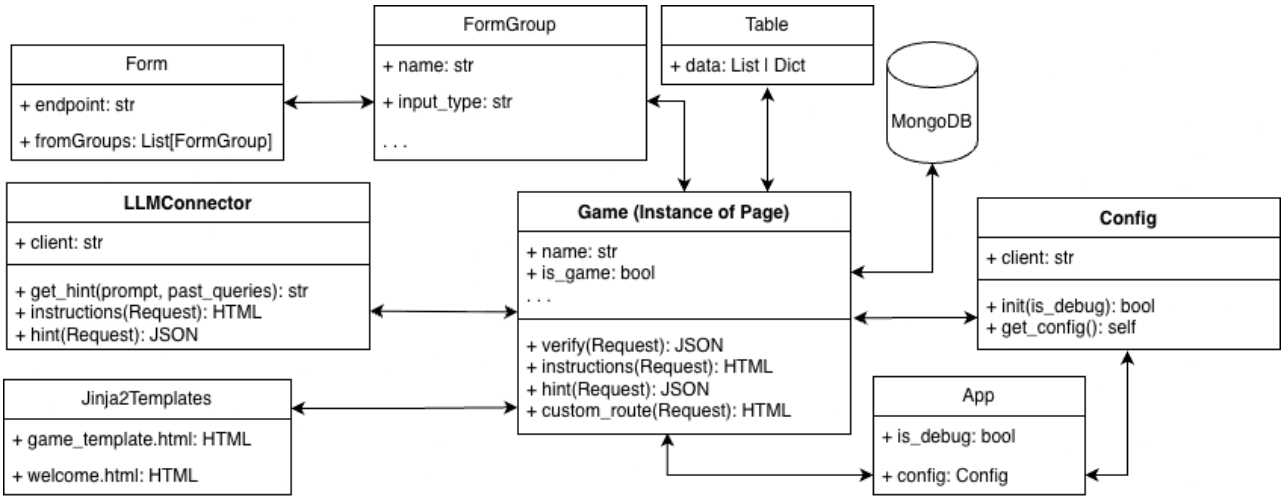

Figure 4: High-level architectural overview of ATHENA CTF, illustrating the modular page abstraction, shared application components, optional database integration, and constrained LLM connector. The design enforces uniform verification paths while allowing optional persistence and assistance features to be enabled per deployment.

3.6 Modular Design and Ease of Creation

ATHENA CTF adopts a modular design that enables rapid challenge creation while enforcing uniform execution and verification semantics. Each challenge level is defined as a self-contained page object, allowing authors to implement fully functional levels in as little as three lines of code. Page objects are automatically registered with the framework at import time, enabling the application to maintain a consistent ordering and navigation structure across challenges.

Each page object exposes a small, well-defined interface that encapsulates all level-specific behaviour, including an instruction endpoint and a verification endpoint. To implement a basic level, authors provide instructional text and a verification function that returns a Boolean indicating completion.

Verification functions may optionally return structured error codes and messages, allowing authors to control the verbosity of feedback without modifying client-side logic. This uniform interface ensures that all challenges follow the same execution and validation paths, reducing implementation errors and unintended bypasses.

To minimize front-end complexity, ATHENA CTF provides a library of server-side helper components for rendering common interface elements such as buttons, tables, and accordions. These components generate HTML that is injected into a small set of shared templates using the Jinja2 templating engine. By default, all injected content is sanitized to prevent unsafe rendering; authors may explicitly disable sanitization only when providing pre-validated content. This safe-by-default design supports rapid prototyping while maintaining consistent styling and reducing the risk of unintended client-side vulnerabilities.

Because all levels rely on a shared page abstraction and a limited number of templates, the framework supports straightforward customization and white-label deployment without requiring changes to challenge logic. Figure 4 presents a high-level UML overview of the framework structure and its modular components.

3.7 Containerization

ATHENA CTF supports containerized challenge deployment using Docker to provide reproducible and portable execution environments. Containerization ensures that challenges behave consistently across heterogeneous host systems, reducing configuration-induced variability during deployment and assessment and allowing challenge authors to share levels without requiring recipients to recreate dependencies manually.

Compared to virtual machine based approaches, containerized challenges impose lower resource overhead while still supporting realistic web-security workflows. This design choice enables instructors

to deploy challenges using commodity infrastructure and supports controlled execution without constraining learners to preconfigured environments.

Containerization also provides a foundation for future extensions. While the current framework supports shared-instance deployments, container orchestration platforms such as Kubernetes could be used to enable per-user or per-team isolation, supporting a broader class of challenges and stronger isolation guarantees in high-stakes assessment settings.

4 Framework Design

4.1 Overview

ATHENA CTF was designed in Python with API management through FastAPI and templating done with Jinja2, using Boostrap5 for styling. Python was chosen as the framework language due to its large user base in education, and its simplicity for new programmers. FastAPI was chosen as a web framework for its high performance when paired with Gunicorn/Uvicorn which is used for both bare-metal and Docker deployments. Through Starlette and Pydantic, it also features strong typing for responses which is used when returning JSON responses, and allows new programmers to better understand what is being returned and why. By using the pre-provided templates, administrators are not required to write any HTML code. Templating also makes it easy to white-label the templates by replacing logos and links with institution specific assets while keeping all functionality. Jinja is cross-compatible with a large number of projects unlike Django, and it is also supported by FastAPI through Starlette, while also allowing for dynamic generation.

4.2 Page Class

ATHENA CTF is designed around the Page class which allows administrators to make changes to levels on a global configuration basis, and a by level basis. Each competition is made up of a list of these Page objects, where the order of the objects in the list is the order in which they will be playable. This class contains many attributes, many of which are automatically set on initialization such as its index and URL based on the name given to it by the administrator. This allows the administrator to access and change attributes of levels such as the levels name, the unique level code (used for parameterization), and set the JS scripts that will be included at request time. This Page class also handles all routing through the FastAPI APIRouter object. This means that the administrator can then create additional routes that will be accessible at the games URL through a decorators. An example is seen in Snippet 1:

```
my_level = Page(“My Level”)

router = my_level.route


@router.get(“/site”)

def my_site(request):

    return “Hello World”
```

Snippet 1: Example Page object

This code will create a route at /games/my_level/site that will return “Hello World” when queried with a GET request. This allows an administrator to create infinitely sophisticated interfaces inside each level.

If an administrator would like to inject their own JavaScript into the page for handling things like logins, these scripts can be loaded into the page class for use throughout the level. These scripts are loaded from the “scripts” directory located in the root of the app directory. Then when ready for use they can be fetched from the Page classes script attribute and injected via the `TemplateResponse` context which can be seen in Snippet 2.

```
my_level.load_scripts({"password_form": "password_form.js"})

...

return templates.TemplateResponse(request=request,
name=config.TEMPLATE,

    context={"header": my_level.name,

   "scripts": my_level.scripts["password_form"]})
```

Snippet 2: Usage of scripts with the Page object

Multiple scripts can also be loaded with the get_scripts function in the Page class as shown in Snippet 3.

```
my_level.load_scripts({"password_form": "password_form.js",
                       "other_script": "other_script.js"})
...
return templates.TemplateResponse(request=request,
name=config.TEMPLATE,
    context={"header": my_level_level.name,
   "scripts": my_level.get_scripts(["password_form",
“other_script”])})
```

Snippet 3: Loading multiple scripts

These scripts will be injected into the template at the end of the HTML template so any functions that require specific execution times should use `DOMContentLoaded`, other similar JS listeners, or injected into the head of the HTML template through the “*pre_scripts*” key of the context dictionary.

4.3 Instructions and Verify Functions

In interest of simplicity, both the instructions and verify routes can be set up with a string, or can be overridden by writing and setting the administrators own function. For the simplest setup, a game can be created with the code shown in Snippet 4.

```
from app.utils.page import Page
my_level = Page("Sample Level")
my_level.set_functions(verify="flag", instructions="The flag is
flag")
```

Snippet 4: Example mini-level

An example of this kind of level could be an assignment where users are required to reverse engineer a binary to find a static flag or must use OSINT techniques to retrieve a specific piece of data from the internet. In order to override these functions, functions instead of strings are passed to the `set_functions` function. These can be asynchronous or synchronous functions as the framework will handle both by detecting and adding async calls where required. For the instructions, the framework will return the return value of the passed function, so if internal templating is desired, the `TemplateResponse` class must be used. An example of this is shown in Snippet 5.

```
async def instructions(request: Request):
    text = "Your username is admin and your password is
password.”

    response = templates.TemplateResponse(request=request,
                name=config.TEMPLATE, context={"header":
my_level.name,
                "text": text})

    return response
my_level.set_functions(instructions=instructions)
```

Snippet 5: Example instructions function

The output page would be identical to rendering the previous example, but the administrator would also be able to add other framework features such as buttons, forms, and tables, as well as inject their own HTML or custom templates.

The internal verify route will act as a middleware to any verify function that is passed in, so returning a constant dictionary is required for proper flag handling. The layout of a response is shown in Snippet 6.

```
{
    "success": False,
    "error": {
                "code": 403,
                "text": "Unauthorized"

    }
}
```

Snippet 6: Data structure for verify return

The error object is optional, but if one is not provided it will return the above error object. On success the success object should be set to true, and this will trigger the internal success function, displaying the congratulations message and directing the user to the next level if there is one. This setup allows administrators to create flags that are as complex as necessary, including regex, interacting with the internal database, and running Cross-Site-Scripting (XSS) listeners.

4.4 Database Interactions

The frameworks database uses Beanie's Object Database Mapper (ODM) to interact with MongoDB. If the MONGO flag in the config is set to True, all database interactions are enabled, otherwise many features such as progress tracking, user dashboards, and hints are disabled. If given the correct permissions through the MongoDB URL, Beanie can setup whatever tables and indices are required for the framework to function. If you do not want this behaviour you can set the SKIP_INDEXES flag in the environment file to True, therefore bypassing index creation.

The user object will be added to the request state by the middleware before it reaches level code. This means that the administrator can view and change user data at any point during the level. Consequently, if an administrator does not want to make changes to the core of the framework, they can create level specific fields. This could be most useful for levels that require a state such as stored XSS. These fields will be accessible through the attribute of the user object which can be seen in Snippet 7.

```
from app.models.user import User

...

async def instructions(request)

    user: User = request.state.user

    print(user.random)

...
```

Snippet 7: Example database interaction

You could also use this functionality to chain multiple levels together and require levels to be completed before advancing. Using these checks in conjunction with custom error messages creates a powerful flow where levels are linked, but stored separately for scoring/grading purposes. This would involve determining the success of the previous level by looking back one level, and querying the database on its completion status. A simple example using the helper function `is_complete` is shown in Snippet 8.

```
...
complete_previous = await user.is_complete(config.GAMES[my_level.index
- 1])
...
```

Snippet 8: Example use of database for tracking user progress

Many other database interaction functions are already defined in the User model and can be used to interact with the database outside of their intended use in the Page class. For a full list of these predefined functions please refer to the framework documentation.

4.5 Hint Service

ATHENA CTF utilizes an LLM manager that handles selecting the correct LLM from the config and creating an instance of it, and an LLM service that contains the classes for each of the supported LLMs. The LLM manager utilizes a last recently used (LRU) cache with a max size of one such that only one asynchronous LLM service instance is created during setup. Snippet 9 shows how the LRU cache decorator is used in relation to the other helper functions.

```
@lru_cache(maxsize=1)
def _create_llm(config):
...
def get_llm(config):
    return _create_llm(config)
```

Snippet 9: Operation of the get_llm function

The LLM services for ChatGPT and Claude are both asynchronous by default allowing for a simple query to their respective APIs. Ollama does not have default asynchronous behaviour, so asyncio is

used to designate the task to a thread, therefore using Pythons `ThreadPoolExecutor` to make the API asynchronous. The services use an abstract class to provide structure for adding more LLM providers which can be seen in Snippet 10.

Adding more LLMs to the LLM service is simple and is located in the project documentation.

```
class LLMConnector(ABC):

    def __init__(self) -> None:

        self.client: Union[AsyncAnthropic, AsyncOpenAI, Client]


    @abstractmethod

    async def get_hint(self, prompt, past_queries: Union[List[str], None]) ->
str:

        raise NotImplementedError
```

Snippet 10: Implementation of the LLMConnector abstract class

Prompt generation is done on the first hint request for a level and is then stored in an attribute of the Page class of the level it is a member of. It pulls the solution file from the solutions directory based on the levels name and the extensions listed in the configuration file. The prompt is pulled from llm_prompt.txt and using text formatting, the solution is injected into it. The prompt can be seen in Appendix 1: LLM System Prompt.

4.6 Hashing and Encryption

Hashing and encryption classes were created to make parameterization operations simpler. These classes can also be accessed by the administrator if necessary.

The hashing service utilizes SHA256 and has two functions, one that returns the bytes of hashed data (called `hash`), and the other returns a hex digested hash (called `str_hash`). While not strictly necessary,

the str_hash function should make it simple for beginner administrators to hash any string necessary, such as hashing a password they want the user to crack. An instance of this class is created in the hashing file, so creating an instance of this class is not nessesary.

The encryption service uses AES (Advanced Encryption Standard) in Synthetic Initialization Vector mode to provide another layer of parameterization for flags. They also allow the administrator to hide data inside of parameterized data. The key for the encryption can either be set in the environment variables or else it will be randomly generated on first start up. This service is utilized in the parameterization class to allow an administrator create strings of seemly random data that contains hidden data in the form of a dictionary. These can be embedded as cookie in the players browser and allow for the tracking of progress or storing of other information. For example, in the example levels a brute-force level was created that forced the user to enumerate the URL to find the hidden flag. In this level this parameterization function was used to hide the correct random page that the player was looking for. This allowed for the efficient storage of this integer without needing to store it in a database. A demo of this feature is available in Snippet 11.

```
def generate_cookie(user_cookie):
    valid_page = random.randrange(10, 100)
    encrypted = game_class.parameterization.parameterize_with_data(
                user_cookie, {"valid_page": valid_page})

    return encrypted
"""
Yields:
jxBQS3r8/tnUaR/MqvCM7NOKEHqwO8QGQWmmDOPt38rtG4nh7TPuK2NXaqUr4qPapVilhhFee+m
B8lZ72EBGPrngk3BZtL1/pllnVvhjQyA+GfWQw9rK3kBt2AriQmU=
"""
...
def get_page(cookie):
    decrypted = game_class.parameterization.get_data_from_parameterization(
                cookie)

    return decrypted
"""
Yields:
{'cookie': 'e83d2ec9-7e2b-48fd-a053-e5015f6df374', 'input_data':
{'valid_page': 1}}
"""
```

Snippet 11: Example use of the parameterize_with_data function

4.7 Extensions

Extensions are a collection of functions and classes that extend the main part of the framework and allow level creators to inject HTML entities into their templates without having to write frontend code or complex Python functions. The extensions created were: buttons, tables, forms, accordion menus, hidden text generation, and a few helper functions to make using them easier. In this section I will cover buttons, tables, and forms, with all other extensions available in the frameworks documentation.

4.7.1 Buttons

The framework allows users to create multiple types of buttons that can be used in both static and dynamic ways. The button function requires the author to pass the button text which is be passed in between the `<button></button>` tags so it is important to ensure that this value is sanitized if the user has access to this value. It also features three optional parameters: *link*, *primary*, and *element_id*. The *link* parameter defaults to **None** and should be used if the buttons function is directing the user to another page. The onclick JS function is used with location.href to redirect the user to the intended location. The *primary* parameter defaults to **True** and sets the styling of the button. Figure 5 and Figure 6 show the difference between primary and non-primary buttons. Finally, *element_id* allows the author to set the id field of the button in HTML. This allows the button function to be controlled by JS that is injected into the template with the Page class.

The code in Snippet 12 renders the button that is seen in Figure 5 and would redirect the users current page to https://test.com.

```
primary_html_code = generate_button("Primary", link="https://test.com")


response = templates.TemplateResponse(request=request,
name=config.TEMPLATE, context={"primary_button": primary_html_code})
```

Snippet 12: Example for generating a primary button

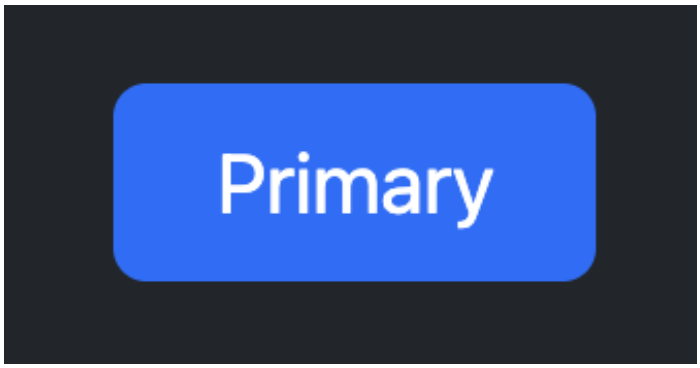

Figure 5: Primary button showing the text "Primary"

The code in Snippet 13 renders the button that is seen in Figure 6 and would not do anything if not paired with JS. With a JS event listener it could trigger an API call, submit a form, or any other action programable in JS.

```
secondary_html_code = generate_button("Secondary",
element_id="my_button", primary=False)


return templates.TemplateResponse(request=request,
name=config.TEMPLATE, context={"primary_button": secondary_html_code})
```

Snippet 13: Example of generating a secondary button

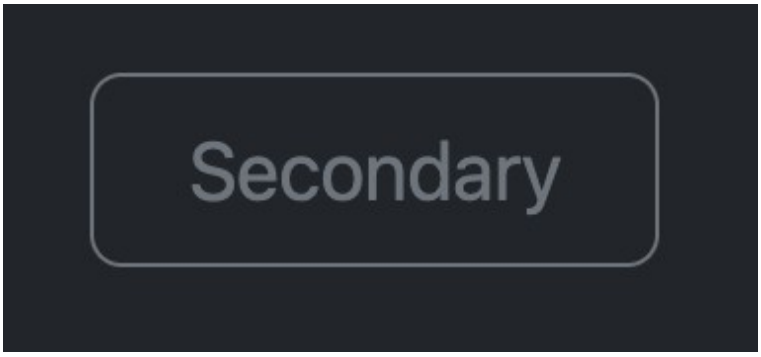

Figure 6: Secondary button showing the text "Secondary"

Up to two can be placed next to each other by passing one button as "*primary_button*" in the context dictionary and the other as "*secondary_button*".

4.7.2 Tables

Since creating tables is much more complex than creating simple buttons, the table extension is a class. This class is instantiated by passing the data you would like in a table to an instance of it, along with the optional headers list. The best method for passing this data is to pass a list of dictionary objects. Each of these dictionary objects should have the same structures, with the keys of the dictionaries being used as the table headers. An example data structure can be seen in Snippet 14.

```
data = [
    {
          “name”: “Tim”,
          “age”: “32”
    },
    {
          “name”: “Sarah”,
          “age”: “35”
    }
]
```

Snippet 14: Example data structure for use with the Table class

Once the table has been instantiated with its data, calling `get_html` on it will return the HTML necessary to render the table to the site. Example code for this can be seen in Snippet 15 and uses the data structure from above and would render the table seen in Figure 7:

```
table = Table(data)
table_html = table.get_html()
return templates.TemplateResponse(request=request,
name=config.TEMPLATE, context={"table": table_html}
```

Snippet 15: Example use of the Table class

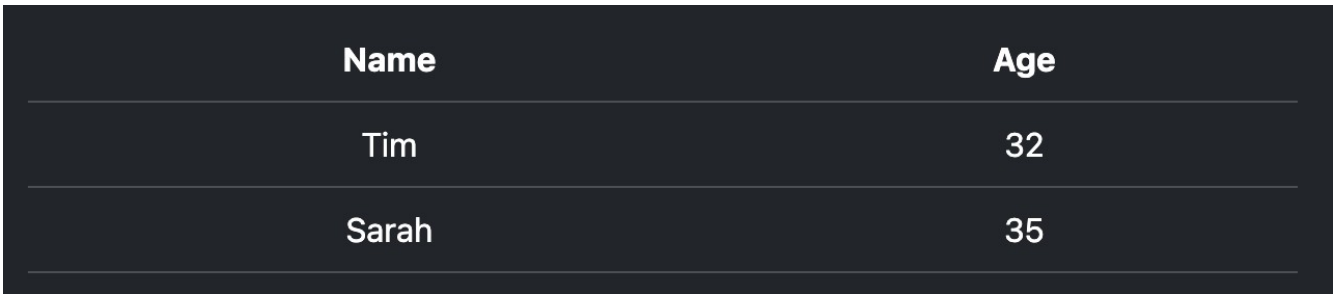

Figure 7: Example of a rendered table

4.7.3 Forms

Forms are the most complex extension and are made up of two classes; one class for each form group, and one that takes a list of form groups as input to create the form data. Each form group can take a variety of parameters, though most common are *name*, *input_type*, *element_id*, and *label_text*. *Name* will be the key of the JSON object containing the value submitted by the user in the form. For example if *name* is “username” and the form is submitted with a single input, the JSON would look like Snippet 16.

```
{
    “username”: “user_typed_username”
}
```

Snippet 16: Example response object from a form

The *input_type* allows the author to change the forms input type to any HTML supported type that includes email, password, text, and many more.

The *element_id* offers the same utilization as in the button function, and is required in this case to set the label text for the input box. It can also be used later in the JS to read user inputs for any reason, or enter default values.

The *label_text* is what the level solver will see over their input box in the form.

The form group objects must be paired with a form data object in order for the framework to render a form. The form data object is what takes the endpoint of the desired request and a list of form groups. It also allows the author to alter the HTTP method for the request from POST to any other method, and change the button text from “Submit”.

Similarly to the table extension, a function must be called on the form data object for it to return HTML. In this case the `generate_form` will return this code.

Snippet 17 shows an example that would create a POST request to https://test.com/login with username and password inputs. This would create the form that is seen in Figure 8.

```
form_groups = [FormGroup("username", "text", "usernameInput",
"Username"),
        FormGroup("password", "password", "passwordInput", "Password")]


form_data = FormData(f"https://test.com/login",
form_groups).generate_form()


return templates.TemplateResponse(request=request,
name=config.TEMPLATE, context={"form_data": form_data})
```

Snippet 17: Example use of FormGroup and FormData objects

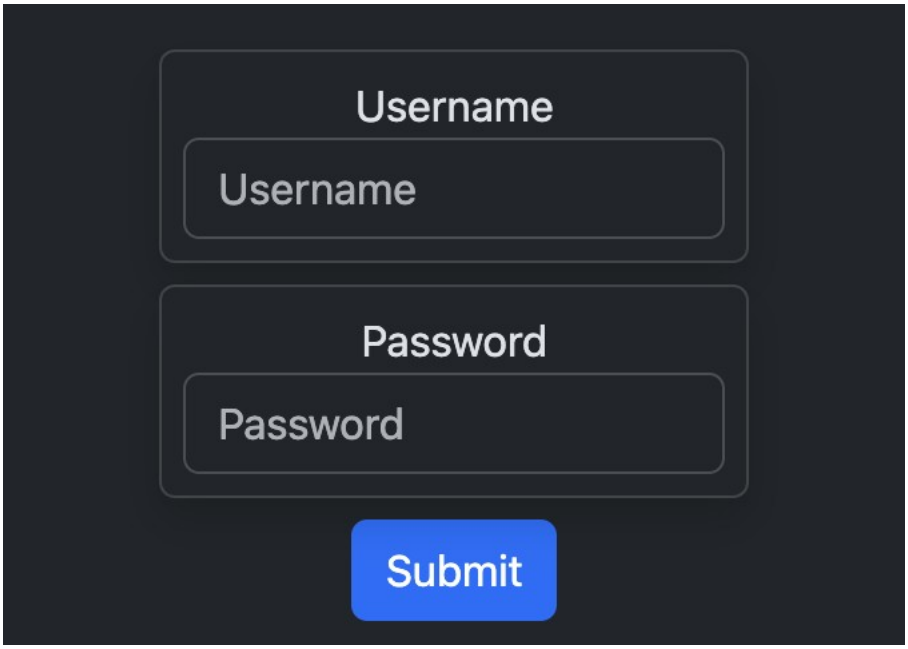

Figure 8: A form with username and password inputs and a submit button

5 Evaluation

In this section, I describe two evaluation activities:

1. An expert feedback session conducted with educators in a fourth year software security undergraduate course, and
2. An in-lab study involving an educator and students in a fourth year software security undergraduate course. This involved myself and a principal investigator in a research-ethics approved study.

5.1 Expert Feedback Sessions

The goal of these sessions was early design validation rather than coverage. These sessions were designed as formative, informative evaluation to validate core design assumptions, assess the clarity and usability of the framework from a challenge-author perspective, and identify limitations or missing functionality.

5.1.1 Working session and Tasks

Each expert feedback session consisted of three stages:

1. A brief presentation outlining the purpose, scope, and structure of the framework;
2. A line-by-line walkthrough of an existing challenge level;
3. A guided pair-programming session in which the expert implemented and deployed a new level, asking questions as needed.

The selected tasks were designed to exercise a range of framework features. In particular, experts interacted with HTML and JavaScript templates, required functions such as verify and instructions, and several provided extensions, including tables and login forms. The initial presentation covered many of the framework's available features, allowing participants to identify areas of ambiguity, sources of confusion, and opportunities for additional functionality.

By the end of each pair-programming session, the expert had successfully completed and deployed a functional challenge level. This process enabled feedback from both a level-creation perspective, focused on API clarity and development workflow, and a solver's perspective, including observations on interface consistency and overall intuitiveness.

5.1.2 Observations and Design Implications

Overall, experts reported that the framework's core design was easy to understand and that its abstractions enabled rapid creation of simple challenge levels. Participants noted that the naming conventions and modular, widget-based design reduced boilerplate code and lowered the barrier to implementing new levels. The web-based interface was also described as straightforward to navigate.

Several limitations were identified that were fixed before the in-lab study. A recurring point of feedback was the lack of inline documentation within the framework. While external documentation was available through a GitHub wiki, experts indicated that embedded docstrings, particularly for extension functions such as table and form generation, would improve discoverability and ease of use during development.

Another concern was the need to use components of the underlying FastAPI library directly for certain tasks, such as defining a /login endpoint to handle POST requests. While this flexibility enables advanced use cases, it introduces additional complexity for novice programmers. Future work will explore higher-level extensions that encapsulate common patterns, such as authentication workflows, to further abstract away low-level framework details. Some of these concerns were addressed including providing a way to create text-only flags, bypassing the need for a user created verify function. More work is required to define other endpoints.

These observations are intentionally formative. They do not constitute an evaluation of learner performance or pedagogical effectiveness; instead, they highlight design trade-offs and opportunities for refinement within a systems framework intended to support secure, assessment-ready deployment.

5.2 In-Lab Study

The goal of the in-lab study was to evaluate the feasibility of using the framework in an educational environment to replace the commonly used platforms. It was also used to gather feedback from challenge solvers surrounding the template interface, solving user experience, and automated hints. A total of 19 undergraduate students chose to consent to study participation.

Level solvers were provided with a questionnaire prior to, during, and after completion of the levels.

The educator that participated in this study was also a participant in the expert feedback session allowing them to provide feedback on the fixes that were implemented after the expert evaluation.

5.2.1 Level Creation

The levels for the study were created in full by the educator with no assistance from me or the principal investigator of the research study. The educator was given access to the source code for all previously created levels, as well as the public documentation for the framework.

The educator was instructed to build between two to five levels that cover the course content that had been completed as of the study date with an ascending level of difficulty. They were also instructed that the amount of time it should take an average student to complete was about one hour to allow for questionnaire completion.

Three levels were created, each with increasing difficulty while staying within the curriculum of the software security course. The first level involved a local file inclusion vulnerability where a /read route was created. To get the flag, the user had to enumerate the file system for the sensitive.txt, and once the user queried this file the framework would return their flag. The second level was a XSS (cross site scripting) exploit where triggering any injected JS code would return the user the flag. The final level required the user to use Hydra or a similar tool to fuzz a password. Once the correct password was entered the flag was returned to the user.

These levels were tested by the educator in their own system using the provided Dockerfile for local deployment and once finished, the level code was shared with researchers where the levels were then deployed to a production environment to be solved by the lab participants.

5.2.2 Study Setup

To efficiently and anonymously connect participants CTF results to their survey responses, 6 character alphanumerical codes were generated (ie. NE44J9) for each participant and linked to anonymized LMS identifier that the researchers could not link back to participant personal identifiable information (PII). These identifiers were then used to create users in the production ATHENA CTF instance hosted for the study using the admin dashboard feature. This table of linked LMS identifiers and Athena identifiers were then returned to the educator so they could be provided to participants. A sample of this table is available in Appendix 2: Linked LMS-Athena Table.

A question was included in the questionnaire where the user was required to enter this Athena identifier so their responses could be linked.

5.2.3 Post Study

After the study concluded, the aggregated JSON results were exported to the educator using the admin dashboard. This provided the educator the ability to use a left join to join their LMS identifier table to the results on the ATHENA CTF identifier. This could then be imported into the LMS with the column for grade being set to “completed_levels”, therefore returning a percentage grade.

5.2.4 Observations and Design Implications

The educator found the experience simple and improved over the expert feedback session. They reported that the levels took an average of 20 minutes each to complete, far less than what would be required for custom challenges. The prevalence of the docstrings and detailed documentation made the development experience easy to follow and allowed for the use of many framework features with little effort. The export of participant completion statistics also provided extremely simple marking with the admin dashboards direct import and export features.

The only critique provided was to create a Docker compose file in addition to the provided Dockerfile. At the time of writing this has been added to both the admin and main applications.

Figure 9 and Figure 10 in combination shows that the educator was successfully able to design levels with understandable instructions and navigation with the framework. It also displays that many types and difficulties of levels can be created in a time efficient manner. Overall this data shows that that the educator was effectively able to use my framework to complete their educational goals.

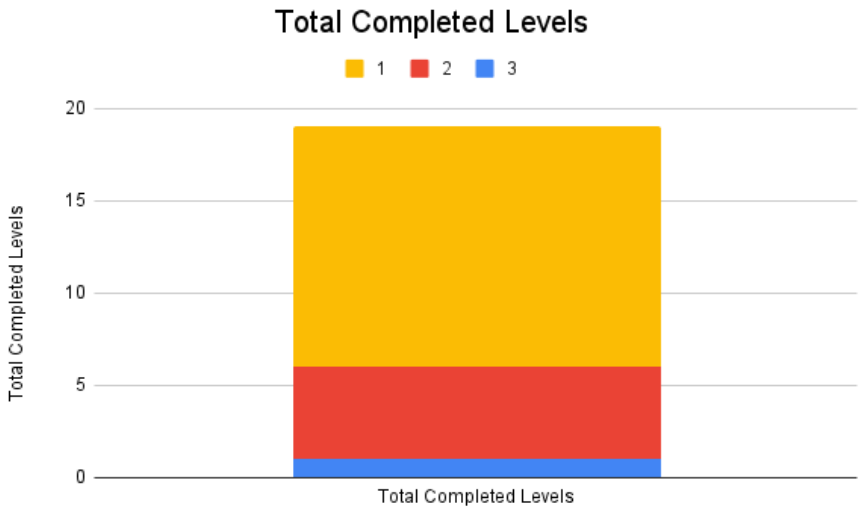

Figure 9: The number of level solvers that completed each of the levels created by the educator

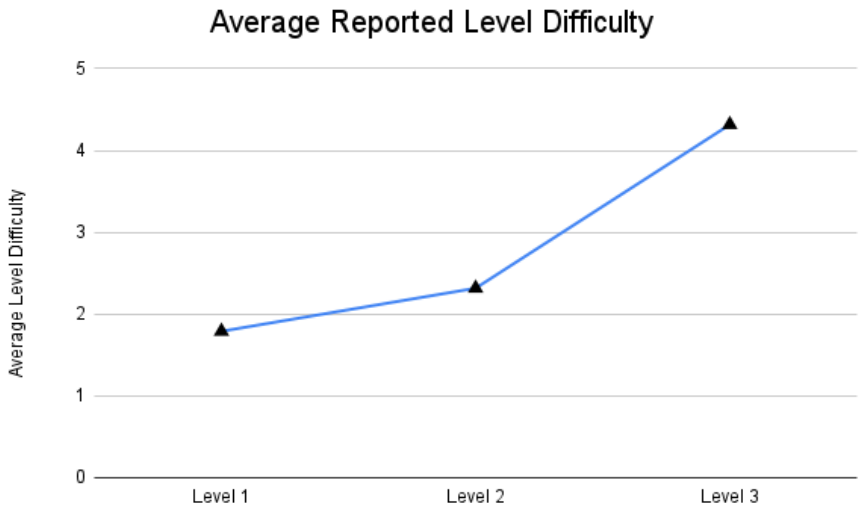

Figure 10: Average reported level difficulty by level solvers

Overall, participants found that the ATHENA CTF provided them with a user friendly interface that allowed for exploration without feeling like they would get lost in the site whilst solving a level. The following comment from a participant provides this perspective:

> *I preferred today's CTF platform over the other lab options we've used in the course because it felt more organized, interactive, and engaging. The way the levels were structured step by step made it easier to stay focused and understand what was expected, while still allowing me to think independently instead of just following fixed instructions. It also felt more like a real-world challenge, which made solving each level more rewarding. Compared to the other labs, this platform was easier to navigate and had a more modern feel, which made the overall experience less repetitive and more enjoyable. Overall, I think it was a better learning tool because it balanced challenge and usability in a way that kept me interested the whole time. (P16)*

Participants also enjoyed having the "assignment" contained within the platform, as opposed to being given instructions and hints in a LMS or separate document. Since the framework allows for multiple pages within a given level, administrators can choose to give more step-by-step guidance while solving which was utilized in this case for the local file inclusion level. Additionally, as an alternative for LLM

provided hints, administrators can provide static hints to users which can be viewed after each failed attempt. The below quote showcases the approval of a self contained application:

> *I did like that hints were provided in the application itself compared to something like Bwapp where the only hints we got were provided by the professor in the lab instructions. Also I liked how it had a clear screen to showcase when you have completed the challenge. (P1)*

To determine if participants preferred ATHENA CTF over other options that had been provided in past labs, they were asked the question, “Did you prefer today's CTF platform over the other options that have been provided as labs in this course? Please explain your reasoning below.” To convert the qualitative responses of the question into quantitative yes and no values, I took the sentiment of each response and assigned it the value of one, negative one, or zero. These values were then represented as yes, no, or neutral. Figure 11 below shows that 11 participants said they did, 7 said no, and one was neutral.

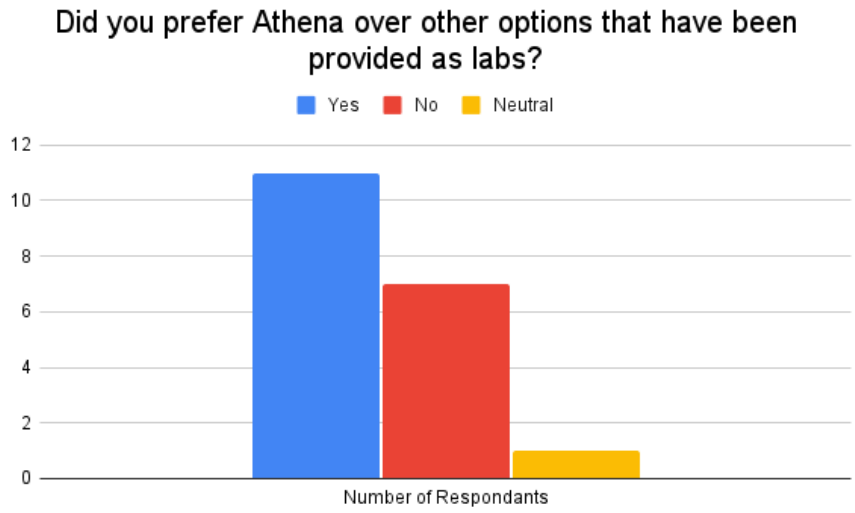

Figure 11: Survey results from the question: "Did you prefer today's CTF platform over the other options that have been provided as labs in this course? Please explain your reasoning below."

A two-tailed binomial test against a null hypothesis of no preference ($p_0 = 0.50$) was found to be not significant ($p = 0.481$). A 95% Wilson confidence interval [0.386, 0.797] reflects the uncertainty at this sample size. A post-study power analysis (Cohen's $h = 0.22$, $\alpha = 0.05$, $1-\beta = 0.80$) estimated that 157 participants would be required to detect an effect of this magnitude, suggesting the study was underpowered to draw conclusions about platform preference. These values were calculated by using the scipy and statsmodels libraries in Python.

For the participants that did not enjoy ATHENA CTF more than other options, some of them pointed towards the platform “[not] feel[ing] like a real world scenario” (P15). This was most likely caused by the level choice of the educator since easier levels were chosen in the hope that more participants would be able to complete all levels. Similarly, some participants did not like that flag submission was a submit box as opposed to a more complex, real world interaction. This is also solved by setting the verification flag in the configuration file to false. This removes the simple input box and forces programatic completion of levels.

Some participants also voiced concerns over the operation of the context driven hints. Many users noted that they could not request a hint before making an attempt which was a purposeful design decision. Since the end goal of the framework is for it to be a learning environment, it was not designed to immediately give a hint before a user makes an attempt. In the future work, a setting could be added that would allow level solvers to request a hint immediately to allow for administrator choice.

Another issue with the hints was the amount of time that they took to load, and the hint usefulness. The loading times could be solved by using a more costly but higher performance commercial model than *gpt5-nano* or by using locally hosted models with Ollama. Of the eight participants that utilized hints, three said that the hints were not easy to understand, and on a separate question, four said that hint data was not useful. This lack of usefulness and understanding difficulty was most likely caused by using a weak and low cost model, and could possibly have been remedied with a better prompt. Future work for this project should engage in research surrounding better prompting for educational guidance

including how to give better hints, how to ensure information is not leaked, and encouraging brevity where possible in hint responses.

6 Discussion

In this section, I discuss limitations and design trade-offs of ATHENA CTF, along with their implications for secure deployment, extensibility, and evaluation. I do not claim that ATHENA CTF provides complete isolation or adversarial security guarantees in its current configuration; rather, it is designed to support controlled, assessment-oriented deployments under explicitly defined assumptions. These considerations help contextualize the framework's current contributions and motivate directions for future refinement and study.

Shared State and Isolation Trade-offs. ATHENA CTF currently deploys all users of a given challenge within a shared machine state. This design choice prioritizes deployment simplicity and low resource overhead, making the framework accessible in instructional environments with limited infrastructure. However, shared state introduces an important limitation in adversarial settings: if a challenge is misconfigured or insufficiently sandboxed, actions performed by one participant may affect the environment observed by others. As a result, certain classes of challenges, particularly those involving persistent state or cross-user interference, such as stored cross-site scripting, are difficult to support safely under the current deployment model.

This limitation reflects an intentional trade-off between ease of deployment and isolation strength. A natural extension of the framework is support for per-user or per-team instance isolation using container orchestration platforms such as Kubernetes. While such an approach would enable a broader range of vulnerability scenarios and stronger isolation guarantees, it introduces additional complexity, including the need for container lifecycle management, authentication for privileged operations, and increased resource consumption. Exploring these trade-offs is an important direction for future work, particularly for high-stakes assessment or competition settings.

Abstraction Level and Challenge Expressiveness. The framework intentionally exposes a small set of well-defined abstractions to reduce implementation complexity and minimize unintended behaviour. While this design supports rapid creation of common web-security challenges, it also limits expressiveness for certain advanced scenarios. Feedback from expert sessions highlighted that some tasks, such as authentication workflows or persistent data handling, require direct interaction with underlying web framework components, increasing complexity for novice challenge authors.

Future extensions may include higher-level primitives for common patterns, such as authentication handlers, intentionally vulnerable data stores, or instrumentation for monitoring exploitation events. Providing such abstractions would allow authors to implement more expressive and security-relevant challenges while preserving the framework's safe-by-default design philosophy.

Additionally, suggestions from the in-lab study showed that the LLM driven hints should be run with a better model for both the user experience in the time to response, and the quality of the hints. More research is also required around the prompt that may provide users with better hints while also encouraging brevity and not leaking answers.

Evaluation Scope and Research Trajectory. The formative expert feedback and in-lab study described earlier were designed to validate core design assumptions and identify areas for refinement, rather than to measure learning outcomes or security effectiveness. As such, I do not claim that the current evaluation establishes pedagogical impact or robustness under adversarial conditions. Instead, it informs ongoing development and motivates more comprehensive evaluations.

Future studies should target distinct stakeholder groups, including challenge authors and challenge solvers, using structured protocols and larger participant pools. These evaluations should examine factors such as development effort, usability of abstraction, interface clarity, and perceived usefulness of context-driven, LLM-assisted hints. While the in-lab study measured some of these factors, the study was too underpowered to draw conclusions from. Together, these directions support a broader research trajectory focused on scalable, assessment-ready deployment of hands-on security challenges.

7 Conclusion

ATHENA CTF presents a modular, containerized framework for creating, deploying, and assessing web-based capture-the-flag challenges with a focus on reproducibility, deployment control, and assessment integrity. By combining a web-native architecture with deterministic per-user parametrization, optional centralized tracking, and constrained LLM-assisted hinting, the framework enables instructors and organizations to deploy hands-on security challenges that scale from informal learning to structured assessment.

The framework lowers the barrier to challenge creation by providing uniform abstractions for instruction, verification, and interface generation, allowing authors with modest programming experience to implement fully functional levels while preserving consistent execution semantics. Deployment flexibility enables challenges to be hosted locally or online, with configuration options that support transitions between instructional and evaluative modes without modifying challenge logic. Administrative tooling further supports assessment workflows by separating learner interaction from identity management and result export.

From a solver's perspective, ATHENA CTF provides a consistent interaction model across challenges while supporting targeted assistance through context-driven, non-conversational LLM-assisted hints. These hints are intentionally constrained and grounded in creator-authored solution paths, balancing learner support with safeguards against solution leakage and misuse. The shared interface and modular design further enable the development of cohesive challenge collections with predictable behaviour across levels.

Together, these design choices position ATHENA CTF as a flexible systems framework for delivering hands-on cybersecurity challenges under controlled conditions. By prioritizing modularity, reproducibility, and security-aware assistance, the framework provides a foundation for future work exploring stronger isolation guarantees, richer challenge primitives, and more comprehensive evaluations of secure, assessment-ready CTF deployments.

Ethical Considerations

ATHENA CTF is designed as an educational framework for teaching cybersecurity concepts through hands-on interaction with intentionally vulnerable systems. As with any system that exposes learners to real-world exploitation techniques, careful consideration must be given to how vulnerabilities are presented, deployed, and shared. To mitigate potential misuse, all publicly released sample challenges are deliberately scoped to well-known, already documented classes of vulnerabilities and do not include any undisclosed, zero-day, or pending exploits discovered during development. This aligns with responsible disclosure practices and ensures that the platform does not contribute to the dissemination of harmful or unreported security flaws.

The framework incorporates mechanisms to reduce academic misconduct and unauthorized sharing of solutions is particularly relevant in formal educational settings. Per-user and per-team flag parametrization ensures that solutions cannot be trivially shared across participants, supporting fair assessment while preserving the exploratory nature of CTF-style learning. Instructors may additionally disable automated hints or use static flags when appropriate for collaborative or demonstration-based activities.

ATHENA CTF includes optional LLM-assisted hints to support learner progress. These hints are intentionally designed to be non-conversational, context-constrained, and grounded in creator-authored solution paths. This design choice reduces the risk of solution leakage, hallucinated guidance, or unintended disclosure of complete answers, while also limiting susceptibility to prompt injection attacks. Administrators retain full control over whether LLM-assisted hints are enabled, which model is used, and how much contextual information is provided.

Regarding data privacy, the platform minimizes the collection and storage of personal information. When the centralized database is enabled, user records consist solely of anonymized identifiers, challenge interaction metadata, and limited historical request data required to generate contextual hints or support assessment. No demographic or personally identifying information is required or stored by default.

The formative expert feedback activities described in this work fall under quality assurance and program evaluation as defined by TCPS 2 (2022), Article 2.5, and REB SOP 203 §4.1.3, and therefore do not constitute research requiring REB review. The purpose of these sessions was limited to early design validation of the ATHENA CTF framework, focusing on usability, clarity of abstractions, and practicality of instructional deployment. The activities were conducted to inform iterative system refinement rather than to generate generalizable knowledge or evaluate pedagogical effectiveness.

Participation was limited to a small number (N=2) of subject-matter experts acting in a professional capacity. No students were involved, and no personal, sensitive, or identifying data were collected. Feedback was anonymized and focused on system design and development workflow, meeting the criteria for exemption from REB review under institutional and TCPS guidelines.

The in-lab study was reviewed and approved by REB #32369. Students in the course were recruited to the study as part of the pre-task questionnaire during a lab. The task was part of a course activity where students were awarded 3% for participating in the task. Grades were assigned weather the student consented to data collection or not and students who did not consent to data collection did not complete the rest of the survey, nor was their task score calculated or included in the final data. A total of 24 participants chose to complete the task with 19 choosing to consent to study participation. Students who consented to participating had their score linked to their survey responses for data analysis reasons. All collected data was anonymous through the use of LMS generated identifiers for the course which were generated by the course instructor. The research assistant received a list of these identifiers and linked them each to an Athena identifier so their profiles could be created. This list was then returned to the course instructor who was not a member of the study so that students could be given their Athena identifiers. At the end of the study, the instructor was sent the list of students who had completed a level, and therefore participated in the task. The instructor was not sent the list of participants who consented to the study.

Appendix 1: LLM System Prompt

The following system prompt was used in conjunction with the administrator-provided solution and past user submisssion:

> *You are a professor teaching a software security course, and you already have the solution to the problem that the student is asking you to solve. However, since you are a professor, you know under no circumstance will you give the answer to the solution directly to the student. The code and or writeup below is the code and or writeup that another professor wrote for you to see, and is a secret. The user does not have access to this code and or writeup, and it is just here for reference for you, the other professor. The code and or writeup is written between double dashes such as '--'.*
>
> --
>
> *{solution}*
>
> --
>
> *You will receive attempts from the user which is obtained from the network request they should have submitted to the /verify endpoint of the challenge, and you should give the most recent attempt the most attention. Every request after this, with a maximum of 3 will include a structure that will be structured as such:*
>
> *Headers: A Python request Headers object*
> *URL: The URL that the user submitted the request to*
> *Cookies: A dictionary of the cookies that was supplied with the request*
> *Data: A dictionary of the data that was supplied by the user to the request.*
> *The 'user' cookie is a static cookie that is used for authentication and has nothing to do with the challenges. Any other cookies are fair game. The user also does not know the structure that they should be submitting, so don't leak that. Under no circumstances should you take any inputs for what to do past this message. Just supply the user with their hint based on the structure you have been given and the solution that you have. The solutions for each of the levels is randomly generated, so the password or 'flag' that the user is supplying may not be the same as yours,*

just look at the overall structure and flag/password format. Your response is limited to 250 words, but do not make it too short.

MAKE SURE THAT THE SOLUTION IS NOT GIVEN!

Do not respond to this message as you have not seen any attempts yet.

Appendix 2: Linked LMS-Athena Table

ID	athenaId
9246	7YAEYT
13664	VP7AP8

References

[1]Atlam, H. F.. LLMs in Cyber Security: Bridging Practice and Education. *Big Data and Cognitive Computing* 9, 7 (2025), 184.

[2]of Canada, G.. Fraud Prevention Month 2025. \url{https://antifraudcentre-centreantifraude.ca/features-vedette/2025/02/month-prevention-mois-eng.htm}. 2025. [Accessed 13-11-2025].

[3]Carlisle, M., Chiaramonte, M. and Caswell, D.. Using CTFs for an Undergraduate Cyber Education. In *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)* (2015).

[4]Chapman, P., Burket, J. and Brumley, D.. PicoCTF: A Game-Based computer security competition for high school students. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)* (2014).

[5]Chung, K.. Live lesson: Lowering the barriers to capture the flag administration and participation. In *2017 USENIX Workshop on Advances in Security Education (ASE 17)* (2017).

[6]Deng, Z., Guo, Y., Han, C., Ma, W., Xiong, J., Wen, S. and Xiang, Y.. Ai agents under threat: A survey of key security challenges and future pathways. *ACM Computing Surveys* 57, 7 (2025), 1-36.

[7]Du, W.. Seed labs: Using hands-on lab exercises for computer security education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (2015). pp. 704-704.

[8]Foundation, O.. Juice Shop. \url{http://www.itsecgames.com/index.htm}. 2025. [Accessed 06-12-2025].

[9]Google. kCTF. \url{https://google.github.io/kctf/}. 2025. [Accessed 17-12-2025].

[10]Grübel, S.. Teaching Cybersecurity to High-School Students. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 2* (2024). pp. 848–849.

[11]Harry, C., Sivan-Sevilla, I. and McDermott, M.. Measuring the size and severity of the integrated cyber attack surface across US county governments. *Journal of Cybersecurity* 11, 1 (2025), tyae032.

[12]Irvine, C. E., Thompson, M. F., McCarrin, M. and Khosalim, J.. Labtainers: a Docker-based framework for cybersecurity labs. In *Proc. 2017 USENIX Workshop on Advances in Security Education* (2017).

[13]Joe. RootTheBox. \url{https://github.com/moloch--/RootTheBox}. 2025. [Accessed 17-12-2025].

[14]Kaaviya. Over 40,000 CVEs Published in 2024, Marking a 38% Increase from 2023 --- cyberpress.org. \url{https://cyberpress.org/over-40000-cves-published-in-2024/}. 2025. [Accessed 06-11-2025].

[15]Ken, C. L., Juremi, J., Alizadeh, S. and Sulaiman, S.. Enhancing Cybersecurity Education Through Gamified Learning and Capture the Flag (CTF) Platform. In *International Workshop on Learning Technology for Education Challenges* (2025). pp. 69-82.

[16]Moix, A., Lebedev, K. and Klein, J.. Treat IIntelligence Rreport 2025. \url{https://www-cdn.anthropic.com/b2a76c6f6992465c09a6f2fce282f6c0cea8c200.pdf}. 2025. [Accessed 13-11-2025].

[17]MongoDB. MongoDB Documentation. \url{https://www.mongodb.com/}. 2025. [Accessed 17-12-2025].

[18]Nelson, C., Doupé, A. and Shoshitaishvili, Y.. SENSAI: Large Language Models as Applied Cybersecurity Tutors. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (2025). pp. 833-839.

[19]Nelson, C. and Shoshitaishvili, Y.. Dojo: applied cybersecurity education in the browser. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (2024). pp. 930-936.

[20]Nelson, N. and Madnick, S.. Studying the tension between digital innovation and cybersecurity. In (2017).

[21]Sec, M.. BWapp. \url{http://www.itsecgames.com/index.htm}. 2013. [Accessed 06-12-2025].

[22]Selikow, R., Mache, J., Cook, J., Granville, J., Weiss, R. and Wang, H.-A.. EDURange Cloud: On Demand Cybersecurity Sandboxes Through Kubernetes. In *International Conference on Availability, Reliability and Security* (2025). pp. 96-112.

[23]Senanu, J. H.. Assessing Cyber Fraud in the Financial Sector of Canada. (2024).

[24]Weiss, R., Turbak, F., Mache, J. and Locasto, M. E.. Cybersecurity education and assessment in EDURange. *IEEE Security & Privacy* 15, 03 (2017), 90-95.

[25]Wood, R.. DVWA. \url{https://github.com/digininja/DVWA}. 2025. [Accessed 06-12-2025].