Improving and Evaluating the Effectiveness of Active Learning in Cybersecurity Pedagogy

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Eric Nieters

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Dr. Peter A. H. Peterson

December, 2025

Acknowledgements

I would like to acknowledge my advisor, Dr. Peter A. H. Peterson, for his motivation and creative ideas in this study. His innovative thinking behind this project allowed me to make a positive impact on all of his future computer security students. Additionally, I would like to acknowledge the SPHERE Project for providing a foundation for this research, enabling our students to use our creation to learn about cybersecurity and relevant exploits. Their new, groundbreaking features are revolutionary for the future of cybersecurity, and we could endlessly develop content to train students using our work.

Abstract

Students at the University of Minnesota Duluth have historically faced challenges in the computer security curriculum and lacked supportive feedback mechanisms. The original labs for this course needed modernization and improvements to better serve students. This study addresses that problem by evaluating whether a more structured, guided approach can improve students' understanding and experience in cybersecurity education. Specifically, it compares a traditional lab format—where students work independently toward a final goal using only a lab manual—with a new, segmented format that breaks tasks into smaller, scaffolded steps. The research involved two cohorts: students from 2023–2024 completed traditional labs, while those from 2024–2025 used the revised version and completed pre- and post-quizzes to assess comprehension. The new labs also include modern, quality-of-life improvements, which aim to improve students' learning outcomes. We evaluated 61 students by collecting their feedback responses and assessing their quiz performance. In addition, we compared the feedback from the updated labs with that of a control group of students who completed the outdated labs. Using hypothesis testing, our results demonstrate that the new lab format achieved core instructional goals, kept students engaged, and met the expected learning outcomes. Students responded to 21 questions about the lab's features, and the responses were generally favorable. These findings underscore the importance of effective instructional design and usability in technical education environments.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Computer security is an incredibly important concept that all computer science students should understand. Many students might treat security as a standard university course, but as cybersecurity threats continue to grow, so do security concerns within the industry. Unfortunately, with the internet growing rapidly with information, more students may feel tempted to rely on tools, like ChatGPT, whenever they encounter difficulties. Cheating disrupts the learning process, and should be taken seriously within cybersecurity education. One of the main issues discussed in this study is that students might be inclined to search online for answers instead of working through problems and engaging in hands-on activities. This is a common issue in many university courses, not just those in computer science.

For this study, we will construct a system designed to support cybersecurity education by incorporating features commonly found in online learning platforms for challenging subjects (such as WebAssign or Pearson). Next, we will design and implement additional features intended to enhance students' understanding of cybersecurity concepts. Finally, we will evaluate the participating students to determine whether they have improved their comprehension of the material and whether they perceive the new features as beneficial to their learning experience.

In Chapter 1, we introduce the problem and the study's goals.

In Chapter 2, we provide the background for this study. Specifically, we examine the

current material that is currently provided to cybersecurity students at our university. Then, we will present a new testbed with the new features that are now available to us. We will describe the features of the original labs which we aim to improve. After presenting the original labs in detail, we introduce the new labs, describe their features, and discuss their implementation.

In Chapter 3, we introduce a new set of labs, and describe their new features and structure. The changes to labs will be refer to the previous chapter, demonstrating how the changes are integrated to enhance students' learning. Additionally, methods for testing will be introduced, and the evaluation process for students will be explained.

In Chapter 4, we present the results of this study. The hypothesis testing method will be described and applied in our study. Once hypothesis testing is complete, we discuss the strengths and weaknesses of these labs based on the results.

In Chapter 5, we further discuss the results, addressing challenges faced, how we catered to different cohorts of students, and how we can interpret these results further.

In Chapter 6, we conclude the study, explain its limitations, and suggest future work that other students or researchers might pursue.

1.2 Motivation and Problem Statement

At the University of Minnesota Duluth (UMD), computer security is a course offered to both undergraduate and graduate students. The second half of the course focuses on exploits, which are essential for understanding real-world cybersecurity. One of the primary goals of the course is to teach students to "bake in security" rather than merely "sprinkle it on top" of their programs. Course topics are designed to raise awareness of cybersecurity threats and help students develop practical skills through hands-on labs.

Historically, these labs were implemented using the Deter testbed, a network security experimentation environment. The original labs required students to connect to remote systems via SSH and follow lengthy lab manuals hosted separately in a web browser. While the exercises covered essential security concepts, but followed an extensive rubric that offered little guidance on how to approach the solutions. Furthermore, the labs provided limited interactivity and minimal real-time feedback. Students often relied on their instructor or teaching assistant to assess their progress. Although the Deter-based

labs provided a strong foundation for technical learning, they also presented several significant challenges.

The resulting problems can be summarized as follows:

1. **Lack of Feedback (P1)** – The labs provided no real-time feedback. Students had to complete the entire exercise before receiving any indication of their performance, and even then, understanding where they lost points often required intervention from an instructor or teaching assistant.

2. **Usability Issues (P2)** – The lab environment was not user-friendly. Students interacted with remote machines via SSH while simultaneously managing lab instructions in a web browser. Students were required to "swap in" their experiments whenever they spent a session working on their lab. When labs are not swapped in, they are inaccessible. More on this is described in Section 2.3.1.

3. **Vulnerability to Solution Sharing (P3)** – The static nature of the labs made them vulnerable to being posted online or solved by tools such as generative AI, which undermined the integrity of the learning process. While this is not unique to cybersecurity education, it represents a growing global issue in academia.

4. **Steep Learning Curve (P4)** – The labs did not offer a gradual introduction to security topics. Due to the wide range of student backgrounds, some students found the material too advanced and became discouraged. Others, with more experience, found it insufficiently challenging. The lack of scaffolding made it difficult to tailor the experience to different proficiency levels.

5. **Obsolescence (P5)** – The Deter testbed is being retired. As a result, the original labs needed to be ported to a new platform, or they would soon become unusable.

6. **Effectiveness (P6)** – Despite covering key cybersecurity topics, the existing labs may lack in teaching long-term learning outcomes. Once a lab was completed, students potentially moved on without learning of the many practical scenarios when an exploit could happen. Without built-in lessons that provided opportunities for applied assessment, it remained unclear whether students could effectively translate theoretical exercises into practical cybersecurity skills.

These limitations collectively form the problem statement addressed in this study. In the following section, we outline a redesign of the labs to overcome these issues and improve both accessibility and educational effectiveness. Redesigned activities could provide a better framework for educators to encourage more thorough engagement with the content. [1]

1.2.1 Solutions and Goals

The problems outlined in the previous section (P1–P5) motivated a comprehensive redesign of the cybersecurity labs at the University of Minnesota Duluth (UMD). In response to these problems, we propose a set of solutions guided by four overarching goals: accessibility, usability, engagement, effectiveness, and content clarity. These goals inform every aspect of the redesigned lab structure and are intended to improve both the technical delivery of content and the overall student learning experience. Below, we outline how each proposed solution addresses the specific problems from Section 1.2.

1. **Dynamic Feedback (G1)** - To overcome the lack of real-time feedback in the original labs, the redesigned environment incorporates automated grading scripts, inline checks, and dynamic responses embedded directly within students' lab notebooks. These feedback tools allow students to monitor their progress as they complete each exercise, reducing the need for manual instructor intervention. This solution contributes to both engagement and accessibility, as it allows students to self-correct and learn iteratively regardless of prior experience.

2. **User-Friendly Improvements (G2)** - The new lab interface replaces the former SSH-based, browser-dependent setup with a streamlined, web-accessible environment that integrates lab instructions, embedded terminals, and contextual help. Students no longer need to manually connect through an external terminal when completing their labs; instead, they can perform all tasks directly within a web browser—even on mobile devices (though it's not recommended due to display reasons). These improvements directly support the usability goal by lowering the barrier to entry and reducing the cognitive load associated with the learning process.

3. **Decreased Temptation in Solution Sharing (G3)** - To reduce the risk of solution sharing via generative AI or online postings, we are developing more dynamic and randomized lab content. This includes parameterized inputs, changing lab conditions, and a focus on open-ended problems that emphasize student reasoning over rote solutions. This approach aligns with the goal of engagement, encouraging deeper cognitive involvement and reducing opportunities for academic dishonesty.

4. **Reduced Learning Curve (G4)** - To make the labs more accessible to students with diverse technical backgrounds, the curriculum now follows a scaffolded, tiered structure. Introductory modules introduce key concepts through simplified tasks, while subsequent labs build in complexity. This progression helps less experienced students build confidence and allows more advanced students to engage with challenging material. This strategy supports both accessibility and content clarity, ensuring that the curriculum serves a broad range of learners.

5. **Support (G5)** - The deprecation of the Deter testbed required a migration of the labs to a new platform. To address this, we ported the existing labs to a new cloud-hosted environment using modern infrastructure capable of long-term support and scalability. The new platform ensures continued availability and allows for easier integration of new tools, features, and updates in the future. This transition supports the overarching goal of content clarity, ensuring that the material remains relevant and accessible.

6. **Effectiveness (G6)** – To improve the long-term effectiveness of cybersecurity education, the redesigned lab environment incorporates applied assessments that reinforce key learning objectives. Within each lab, students engage with follow-up questions, learn how to both exploit and remediate vulnerabilities, and participate in large-scale automated simulations that demonstrate how the covered exploits manifest in real-world systems. These additions strengthen conceptual understanding, encourage critical thinking, and help ensure that students retain and can apply their skills beyond the controlled lab setting.

We address these goals by implementing several new features and design improvements, as described in Section 3.1. Notable examples include: checking the state of students' labs to provide dynamic feedback (G1, G2, G4, G5); implementing auto-saving and auto-grading functionality (G2, G5); introducing incremental steps that reduce the cognitive load presented up front (G4, G6); and providing external links to official documentation to support the lab material (G3).

To evaluate the effectiveness of these techniques, we will invite students to participate in the study on a voluntary basis. Consenting participants will complete a pre-quiz and a post-quiz anonymously before and after each lab. Both quizzes contain identical questions that assess concepts expected to be taught through the labs. The labs themselves do not contain "hidden answers" to these quizzes; rather, students are expected to learn the relevant tools and exploits by reading and completing the lab activities before attempting the post-quiz. At the end of the semester, students will complete a survey to rate the features of the redesigned labs on a scale from 1 to 10. Students who previously completed the original, unchanged labs will also be invited to take the same survey. Finally, hypothesis testing will be conducted to determine whether the redesigned labs show statistically significant improvements with the new features.

In conclusion, the proposed solutions directly address the limitations of the original Deter-based labs while aligning with the broader goals of accessibility, usability, engagement, and content clarity. By modernizing the infrastructure, integrating real-time feedback, improving the user interface, introducing dynamic content, and scaffolding the curriculum, the redesigned labs aim to provide a more effective and inclusive learning experience. These changes are not only intended to solve immediate issues, but also to support long-term adaptability in cybersecurity education. The following sections will explore the technical design and educational strategies behind these improvements in greater detail.

Chapter 2

Background

2.1 Modern Pedagogy and Challenges

In this section, we will discuss a little about modern pedagogy, as well as some challenges that are commonly faced by students who take computer science coursework.

2.1.1 Online Resources and Cheating

Before AI became prevalent, computer security students often relied on online platforms like Chegg for homework help. Although Chegg is promoted as a tool to facilitate learning, its use for obtaining direct answers has been associated with diminished comprehension of practical skills. In cybersecurity, where hands-on experience and problem-solving are critical, relying solely on external solutions may lead to a superficial understanding of complex concepts. Students who receive answers without engaging deeply with the material risk missing the development of essential analytical skills and the ability to identify vulnerabilities—a cornerstone of effective cybersecurity practice.

Furthermore, the disruption caused by the COVID-19 pandemic significantly altered students' learning behaviors. The increased reliance on remote learning environments, coupled with the ease of accessing online help, led to a marked rise in the use of Chegg and similar platforms. Figure 2.1 illustrates the annual growth in the number of questions posted on Chegg, underscoring how the pandemic indirectly fueled academic dishonesty. As students struggled to adapt to new modes of instruction, the temptation to use shortcut methods for assignment completion grew, leading to long-term concerns

about knowledge retention and skill mastery. [2]

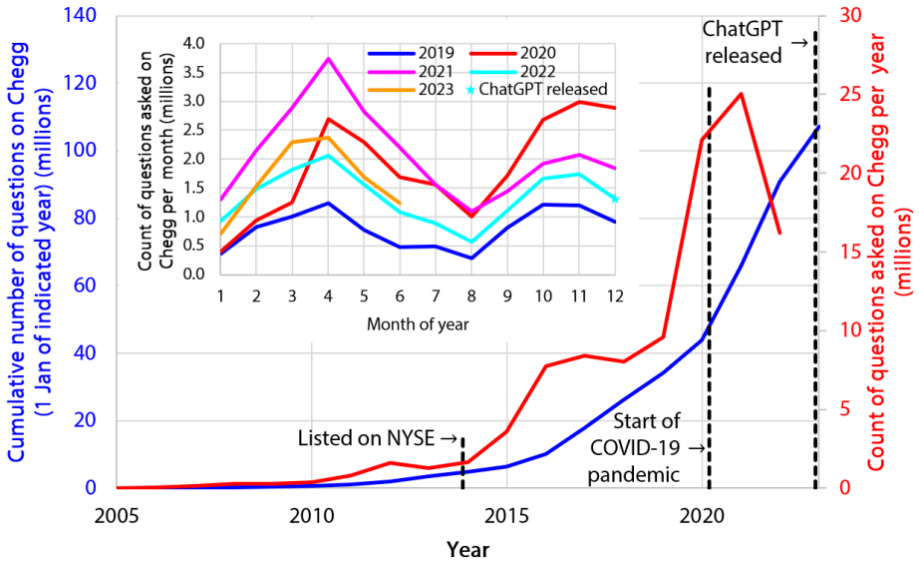

Figure 2.1: Annual Growth Statistics of Questions Asked on Chegg

From the perspective of cybersecurity education, the misuse of online resources represents a paradox. On one hand, such platforms can offer a wealth of information, tutorials, and collaborative problem-solving opportunities that, when used appropriately, enhance learning. On the other hand, the prevalence of answer-sharing undermines the learning process, which is especially detrimental in a field that requires both theoretical understanding and practical competence. The misuse of these resources not only impacts individual student performance but can also have broader implications. For instance, graduates who have relied on external solutions may lack the critical thinking and investigative skills needed in real-world cybersecurity roles, where the ability to analyze and respond to novel threats is paramount.

In response to these challenges, educators in cybersecurity have begun to re-evaluate assessment methods. Alternative approaches such as project-based assessments, open-book exams with real-time monitoring, and collaborative exercises are being considered to better gauge student understanding and to discourage reliance on external shortcuts. [3] In addition, institutions are exploring the integration of academic integrity modules and the use of plagiarism-detection tools tailored to coding and technical assignments.

[4] These initiatives aim to foster a more robust learning environment where students are encouraged to develop their own problem-solving strategies rather than depending on readily available answers.

Ultimately, the experience with platforms like Chegg serves as an important lesson in balancing the benefits of digital resources with the need to maintain rigorous academic standards. For cybersecurity education in particular, the focus must remain on cultivating a deep, hands-on understanding of the subject matter—an essential requirement for securing digital infrastructure in an increasingly complex threat landscape.

2.1.2 Importance Of Cybersecurity Education

In an era where digital infrastructures underpin almost every facet of society, the importance of cybersecurity education cannot be overstated. As organizations increasingly rely on technology, the risks posed by cyber threats continue to grow both in sophistication and scale. This rapidly evolving threat landscape makes it imperative for educational institutions to equip students with the knowledge and skills necessary to defend critical systems and sensitive data.

One of the key reasons cybersecurity education is vital is the continuous emergence of new vulnerabilities and attacks. With cyber-adversaries constantly refining their tactics, techniques, and procedures, a non-interactive curriculum may not suffice. Students must be exposed to current real-world scenarios and practical exercises that simulate live attacks and defense strategies. This hands-on approach not only deepens theoretical understanding, but also fosters critical problem-solving skills that are essential when responding to dynamic threats.

Moreover, as illustrated in the previous subsection, the misuse of online resources—such as relying on platforms like Chegg for easy answers—poses a significant challenge to genuine learning. This issue underscores the broader educational dilemma: while digital resources offer invaluable support, they can also create a false sense of proficiency if not integrated into a rigorous and interactive learning framework. Cybersecurity education must, therefore, strike a delicate balance by leveraging these resources as tools for enrichment rather than shortcuts that compromise skill development.

The importance of cybersecurity education extends beyond individual competence. A well-trained cybersecurity workforce forms the backbone of national security and

economic stability. Organizations, ranging from small businesses to multinational corporations, are increasingly vulnerable to cyberattacks. Graduates who have received a robust education in cybersecurity are better prepared to implement effective security measures, conduct risk assessments, and manage incident responses. Their expertise contributes to creating a safer digital environment, reducing the overall risk of data breaches and service disruptions.

Additionally, the interdisciplinary nature of cybersecurity means that effective education in this field often requires collaboration across different majors. Courses in Computer Science, law, ethics, and policy, for instance, all contribute to a comprehensive understanding of cybersecurity challenges. This approach not only prepares students to deal with technical issues, but also equips them with the perspective needed to navigate the aspects of cybersecurity.

In conclusion, as digital technologies become further integrated into every aspect of modern life, the significance of cybersecurity education grows significantly. Institutions must continuously adapt their curricula to address the emerging threats and ensure that students develop both the technical acumen and critical thinking skills needed to protect and secure our digital future.

2.1.3 Drawbacks to Poor Education

Inadequate or poor education in cybersecurity carries significant long-term consequences for individuals, organizations, and society as a whole. When educational programs fail to provide a solid foundation in both theory and practice, several critical drawbacks can emerge.

First, graduates may lack the depth of understanding and critical problem-solving skills required to effectively anticipate and mitigate cyber threats. This gap in competence can lead to a workforce that is ill-prepared to handle real-world incidents, potentially resulting in increased vulnerability of networks and systems. Without a robust training in ethical hacking, intrusion detection, and incident response, professionals might struggle to diagnose or remedy complex security breaches.

Another major drawback is the potential over-reliance on superficial solutions and shortcuts. As discussed earlier, the misuse of online resources can create a false sense of proficiency. When students depend on readily available answers without engaging with

the underlying principles, they miss out on the opportunity to develop analytical skills and innovative thinking. This not only affects individual learning outcomes, but also impedes the collective progress of the cybersecurity field.

Poor education also raises ethical and professional concerns. A lack of emphasis on academic integrity and ethical considerations can lead to a culture where shortcuts are normalized. Over time, this erosion of ethical standards might encourage behaviors that compromise both personal and organizational security. In high-stakes environments, such as national infrastructure or critical business systems, the consequences of ethical lapses can be severe and far-reaching.

Moreover, inadequate education in cybersecurity can have economic and national security implications. Organizations that hire under prepared professionals may face higher risks of data breaches and operational failures, which in turn can result in significant financial losses and damage to reputation. At the national level, the absence of a well-trained cybersecurity workforce weakens the overall resilience of digital infrastructure against state-sponsored and cybercriminal attacks.

Finally, a deficiency in cybersecurity education can stifle innovation within the field. As technology continues to evolve, a dynamic and well-informed workforce is essential for developing new strategies and technologies to combat emerging threats. When educational programs do not keep pace with industry demands, research and development may slow, leaving the sector less capable of addressing future challenges.

In summary, the drawbacks of poor cybersecurity education are sparse, affecting not only the technical competence and ethical grounding of individuals, but also the broader security, economic stability, and innovative potential of organizations and society.

2.2 Overview of Experiments and Research Design

To address the educational challenges discussed above, this thesis investigates whether redesigned cybersecurity lab exercises can improve student learning outcomes, engagement, and ethical decision-making compared to traditional coursework. The goal is to determine whether a more interactive, modernized approach to cybersecurity education produces measurable gains in comprehension and problem-solving ability.

The project is guided by the following central hypothesis: Students who complete

the redesigned labs will demonstrate significantly higher understanding and retention of cybersecurity concepts, as well as improved ethical reasoning, compared to students who completed the legacy labs.

To test this hypothesis, two main experiments were developed and conducted during the 2023–2024 academic year:

- **Evaluating Educational Effectiveness (E1)** - This experiment investigates whether the newly designed labs themselves are effective at improving comprehension and engagement. Students completed the redesigned labs and then participated in post-lab assessments, surveys, and reflection assignments to measure perceived difficulty, satisfaction, and conceptual understanding.

- **Comparing New vs. Legacy Labs (E2)** - This experiment compares outcomes between two groups: students using the redesigned labs (experimental group) and students who previously completed the legacy labs (control group). The comparison focuses on metrics such as lab performance scores, post-assessment results, and retention tests, in order to evaluate whether the new approach offers a significant pedagogical improvement.

These experiments together provide both a qualitative and quantitative basis for assessing how redesigned, modern cybersecurity labs can address the educational challenges identified in earlier sections. The following section describes the original (legacy) lab exercises used at UMD prior to the redesign.

2.3 Legacy Exercises

Next, we will describe the set of old lab cybersecurity exercises that were originally used at the UMD. We will name these labs as "legacy" exercises, as they were the original labs which were inferred to be less suitable for teaching students about exploits.

As mentioned in Section 1.2.1, we proposed solutions towards our problems being addressed in this study. Our ultimate goal of this study is to present the cybersecurity labs taught at UMD in a modernized format. This includes features that can be found in heavily developed classroom material, such as interactive figures and homework

questions that are found commonly in general STEM courses. We can list two experiments for how this will improve cybersecurity pedagogy, which will be more detailed in Chapter 3.

The seven labs listed below were the original seven labs that were taught at UMD. Students that have used these labs are categorized as the control group, since the newly-designed labs were not ready for distribution at the time. These legacy labs were given from 2015 to 2023, but for this experiment, students who opted-in for this study took the labs during the 2023-2024 school year. Our cohort of students who completed the "legacy exercises" performed them on SPHERE, even though the first lab was titled "Intro to DeterLab". This lab was administered only a few weeks after MergeTB transitioned to SPHERE and prior to the development of our new SPHERE-based exercises.

- **Intro to DeterLab**: An introduction to the UNIX environment, which was also set up to prepare students to work in the DeterLab testbed. DeterLab (cyber DEfense Technology Experimental Research Laboratory) is a computing facility for cybersecurity research, testing, and education. [5] More about DeterLab's architecture will be discussed in Chapter 2.3. This lab consisted of finding five hidden images across a UNIX environment. Students are given documentation about the `find`, `xargs`, and `locate` commands, and they are expected to find these five images using these commands. Little guidance is provided to students for finding these images.

- **POSIX Permissions**: Students are introduced to basic software tools such as `adduser`, `chfn`, `passwd`, `addgroup`, `usermod`, `chown`, `chgrp`, and `chmod`. They are then given step-by-step instructions to perform specific tasks, such as creating directories, groups, and users, and adding users to groups. From there, a "Ballot Box" and "TPS Reports" scenario was given, where students must think about which permissions that they must apply, based on the reading on POSIX permissions that were provided to them. Students can verify their configurations by running a script that automatically tests all permissions and reports whether the scenario was completed correctly. Finally, students answer five short-response questions assessing their comprehension of POSIX permissions, followed by three additional questions evaluating their understanding of potential vulnerabilities

within their configured permission sets.

- **Buffer Overflow**: Students are given an unsafe C program called Wormwood, a nuclear reactor simulator. Before students patch the vulnerabilities in the program, they must figure out what catastrophe can be made be exploiting these functions. Students are expected to find four vulnerabilities in C that pertain to unbounded and unsanitized C functions. Students are given four types of exploits, which may be found and patched in any order. The four vulnerabilities are: buffer overflow, off-by-one error, integer overflow/underflow, and string format vulnerabilities.

- **Pathname Attacks**: Students are provided directions with how pathname attacks work, special pathname tokens, and hard/symbolic links. Next, students are given a demonstration of how pathname canonicalization works, as well as some software tools and HTTP/CGI protocols. For the tasks of the lab, a SUID-root blog post display software is provided to students. Students must find a vulnerability to read `/etc/shadow`, then create a patch in the Perl program, using the `patch` command in Unix.

- **SQL Injection**: SQL examples are provided to students, with some introduction to the syntax and how SQL commands are structured in lecture. Students are not restricted from using online documentation; therefore, those who are not proficient in SQL may refer to online resources. The lab jumps directly into SQL injection after giving three examples of SQL code, showing how SQL code is handled in PHP. Input validation and sanitization is provided as a reminder to students, then shows how SQL injection can occur in a payload. Prepared statements are described, with an example included. Some extra software tools are provided, and the MySQL command is introduced. For the tasks, students are given a vulnerable bank website called FrobozzCo Comunity Credit Union (FCCU). Students must execute various SQL injection commands to achieve specific goals (e.g., gaining access to accounts, transferring money out of an account, etc.). Then, they fix the vulnerability and create a patch against the source. Finally, students must write a memo describing how they found the flaw, fixed it, and how a recovery plan would work.

- **Cross-Site Scripting (XSS)**: Students are given some background on Persistent Cross Site Scripting, and some PHP commands that are used for sanitizing strings. The experiment configuration is provided to students, describing how different servers on this experiment are configured. Instructions for how the website is accessed are given to students, and the task is provided to students. The task instructs students with how to deliver the payload to infected pages. They are also provided four different sanitization functions that increase in difficulty, which they must exploit. They must create a patch file once again, and provide an "attack memo" and a "fix memo", describing how they found this exploit, and how they fixed it.

- **Firewalls**: Stateless and stateful firewalls are described to students, then followed up with firewall policy design. Students are introduced to a standard Linux firewall tool, called `iptables`. Students are also taught how to use a few other tools for analyzing network traffic, like `nmap`, `ifconfig`, `telnet`, and `netcat`. The lab manual provides a table of 11 network scenarios, which they must experiment with different firewall rules and test them across two computers.

It's important to note that the interfaces of these labs are a simple terminal environment. Additionally, some labs have a dedicated save/load script, which generates a tarball (a Unix-friendly zip file) with the lab's resources, and can restore them later. When this tarball is generated, students must use the `scp` command to download their work from the testbed, which can be a learning curve for some students. Some programs, such as WinSCP, can simplify this for students, not requiring them to memorize the command, and simply drag-and-drop their tarball from their experiment to their machine.

2.3.1 Legacy Lab Architecture

The legacy labs used in the UMD computer security course was originally hosted on DeterLab (DETER - DEfense Technology Experimental Research), which was a testbed build for cybersecurity research, testing, and education. DeterLab has been in use since 2003, before the testbed became the NSF-funded alternative, SPHERE (more in

Section 2.4). [6] In this section, we will discuss the original structure of the legacy labs, and what makes them inconvenient.

DeterLab provides two locations for students to work within: DeterLab itself, and their "experimental nodes". A node is a reserved Unix environment, built for testing cybersecurity flaws. This is a disposable machine, as they're allowed to break anything that they want within this environment (within reason of cybersecurity). Their DeterLab home directory is a more restricted location, where they have no `sudo` (or admin) permissions, and are not as free to break their environment. Their DeterLab environment is permanent, as long as their account is active.

Regarding account access, instructors can obtain DeterLab accounts after a manual review process conducted by the system operators. Once an instructor is approved, the instructor may create two different types of accounts: TAs and students. TAs are provided elevated access, where they may do everything that an instructor can do, besides deleting the instructor and other TAs. Temporary student accounts associated with a class are created by the instructor or TA(s) through a web interface, and students receive their assigned username and password through email.

Now, we will describe the general approach students use to perform legacy labs within DeterLab. Students log into DeterLab, where they need to navigate to a menu titled "Experimentation". From there, students click "Begin an Experiment", where they must select the class that they're enrolled in, and provide a description of the experiment that they wish to begin. One feature of DeterLab is to have instructors provide an "NS file". An "NS file", which stands for "Name Server", create the topology of the network, instructing how virtual machines can connect with one another. Students retrieved the NS file for their lab from the lab manual which they wish to begin. Rather than being provided an actual file, students were provided a pathname for this NS file, which was copied and pasted into DeterLab. Some default parameters—such as `idle-swap` and `auto-swap`—are left unchanged. These parameters will be described in further detail later in this section.

After configuring the lab, students click "Swap In Immediately", which tells DeterLab to begin allocating resources for their experiment. This takes 5-10 minutes, and students receive an email once their experiment is "swapped in" (Figure 2.2). Idle-swap is an automatic (or manual) process that releases resources back to DeterLab. When

no activity within the idle-swap period is detected, students will be warned about their experiment being terminated. Auto-swapping is a maximum amount of time that the experiment may continue for. Whether it's active or not. The default time period for idle-swapping is one hour, and the default time period for auto-swapping is six hours.

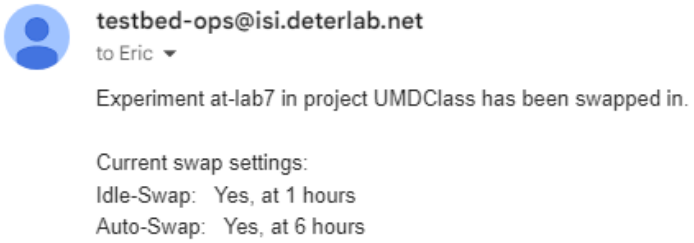

Figure 2.2: Swapping Email Example

One trouble with swapping experiments is that the lab resources borrowed from DeterLab are wiped. When an experiment is swapped out, students are responsible for first saving material onto their home, project, or group directory (persistent Deter storage they can access). Students are provided a permanent storage location for their files, as long as their account is still active. This is a safe location for students to store files, and is usually good for one semester. These accounts are recycled, so that new accounts can be made for a new class.

There is a clear issue with swapping: Students are liable for saving/loading their work. The labs taught at UMD contain scripts for saving and loading their work, but these must be performed manually and regularly. These save scripts were written by Dr. Peter A. H. Peterson (UMD) and his advisor, Dr. Peter Reiher (University of California, Los Angeles), with occasional help from a student, Brandon Paulsen, and another professor, Tanya Crenshaw (University of Portland). When a save script is run, they must use the `cp` or `scp` commands to move their tarball into their DeterLab home directory. Files within this directory are safe from swapping. When it's time to swap an experiment back in, students must use `scp` again to send their file back to the experimental node, then run a load script to reverse their work.

When a lab is complete, students will use a program, like WinSCP, to easily drag-and-drop their tarball to their local machine, then upload the file to Canvas, a common website used for course management at universities. From there, the instructor/TA

will download all students' submissions, and send them to their instructor account on DeterLab. Their work is manually checked within an experimental node, and once their grade is calculated, their grades are individually typed into Canvas. For instructors or teaching assistants, they must unzip each students' tarball, manually check their work, then reset the VM for the next student's work. This was time-consuming, and this issue with grading will be addressed in Section 3.1.10.

These experiments are started within a dated user interface, which can be daunting to students at first. Students are given several options for configuring their experiments, such as assigning it to a project, setting a name, NS file, idle/auto swap times, and more. This information may be more beneficial to researchers, rather than students (Figure 2.3). Once the experiment is started, students are required to open a local terminal on their machine so that it can be accessed. Their DeterLab account is accessed by using the `ssh` command, followed by an argument to connect to the server.

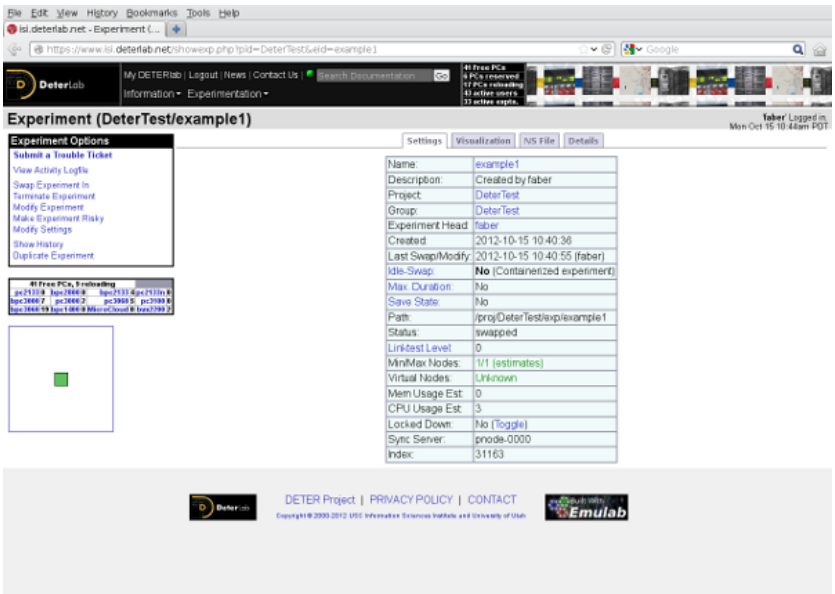

Figure 2.3: DeterLab User Interface From Documentation

For students to connect to DeterLab, the official documentation provides steps with how to access an experimental node using PuTTY, a lightweight SSH client for Windows. This is a user-friendly client for students so that they are not as confused with how SSH is used to access their nodes. Students open PuTTY, and type in a set of values for their

host name and port (Figure 2.4). Students type in their username and password, then type a command to SSH into their experimental node, which is provided to students on DeterLab.

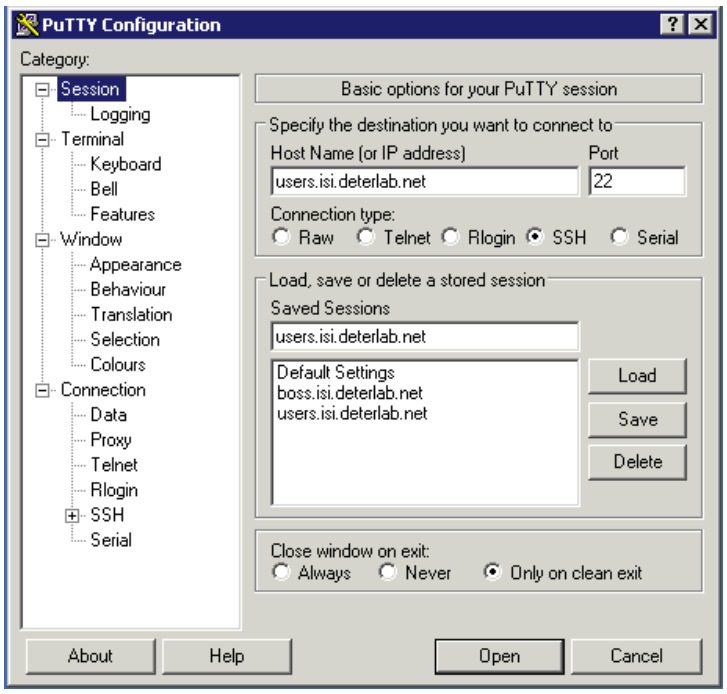

Figure 2.4: PuTTY Configuration for DeterLab

Certain labs, like Pathname, SQL Injection, and Cross-Site Scripting, require web resources to be routed to the student's computer. This introduces the "port forwarding" mechanic. Port forwarding allows resources that are hosted on a remote port to be forwarded to a local port on a machine. (Port forwarding can be thought of like using a VPN to access a campus, except that port forwarding provides only one specific application.) For our cybersecurity labs, we rely on port forwarding so that students may access, on their own machines, vulnerable websites hosted within Deterlab on the student's experimental nodes. Port forwarding requires additional manual setup by the students. Port forwarding can also be done in PuTTY, where students must expand the SSH category in PuTTY, then type the source port and destination (Figure 2.5).

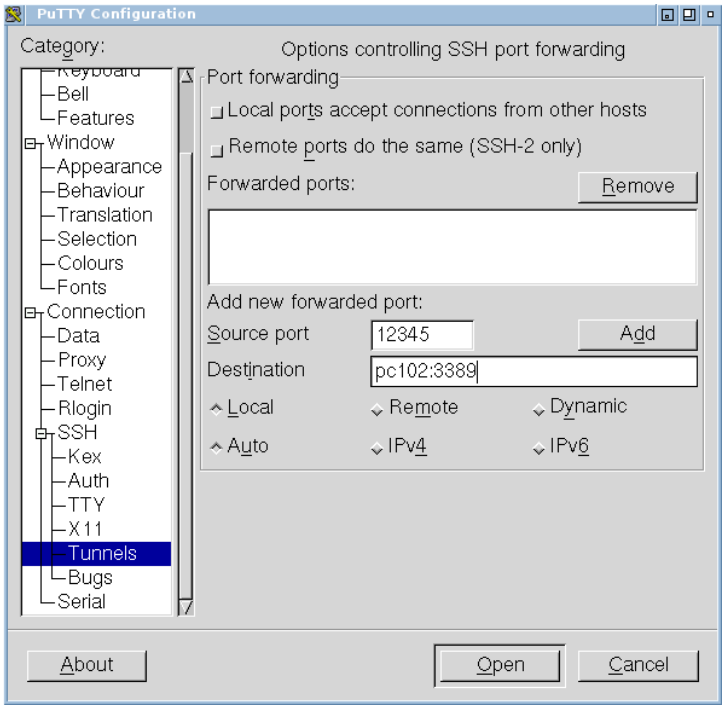

Figure 2.5: DeterLab Port Forwarding

In the diagram, the source port for our lab is port 8080, and the node name is provided as `server`. These are provided to students in the lab manual. Once students use SSH and port forwarding to connect to their node, they may access the website by navigating to http://localhost:8080. When port forwarding is forgotten, students will be unable to view the website for testing.

2.3.2 Weaknesses and Challenges

There are some drawbacks to DeterLab and legacy labs which we aim to address in our new exercises. We will first describe some weaknesses to using DeterLab, and describe how they add challenge to students' learning. Then, we will discuss challenges with the legacy lab's material, and why it may hinder some students' learning experience.

The **user interface** may provide some difficulty to students to navigate their experiments. Loading an experiment isn't a simple button click, as it requires a form to be

filled out, and a pathname to a NS file to be provided to students. The NS file is automatically stored on DeterLab, which provides some convenience to students. However, to find this URL, students are required to scroll past the lab's reading, and navigate to the "Setup" section of their lab. The path is copied, then navigated back to the experiment form, where it's pasted before the experiment may begin.

Saving and loading material is inconvenient for students, as they must rely on manually saving their progress through shell scripts. When their labs are saved, they must navigate from their experimental node to their DeterLab home directory, then use `cp` and `scp` to copy their tarball from their node to their home directory. The reverse is done for loading their labs.

Loading times require students to wait anywhere between 5-10 minutes before a lab can be fully started. This may cause students to forget that their lab is initializing, or may become side-tracked with other activities. Students may lose interest over-time, as loading times may vary, depending on how long their labs take to start. This may be due to server load, maintenance, or the amount of materials needed to start the lab.

A **terminal-only interface** may affect our cohort of students. Some students may be more experienced with navigating a terminal than others. Those students rely on several tutorials, which can slow down their learning experience. This may also affect students across operating systems, as PuTTY is described in the documentation for DeterLab. However, this is for a Windows environment, and students that use MacOS or Linux may rely on other guides with how to connect to DeterLab through a terminal.

Additionally, a terminal interface lacks many of the features that students commonly rely on for basic interaction. Unlike graphical environments, terminals do not allow users to click buttons, open webpages, or perform drag-and-drop actions. All interactions within the lab are conducted exclusively through the command line. Students who are not used to a Unix environment are significantly hindered with basic operations, like navigating into a directory, renaming files, and running a program. For these students, the unfamiliar interface imposes additional cognitive load, compounding the difficulty of the lab tasks themselves.

Automatic swap-outs will wipe a student's progress within six hours, and requires students to remain active in their nodes up to one hour. When a student is attempting to look up a guide or resource, they may spend too much time and forget that their lab

is running. Another scenario is that students may need to step away from their lab, and forget to save, which wipes their progress when nodes are automatically re-allocated back to DeterLab.

Grading faces a similar issue as saving/loading, since tarballs must be obtained from students, an experimental node is created for grading, the tarballs are uploaded to the node, then tested. Once tested, the node is reset (ideally), and the next student is graded. This is an inefficient way to grade students' work.

Limited hands-on demonstration does not engage students, as they are provided several paragraphs worth of text with few examples. Each lab provides a breakdown for what the objectives are, and students will have to experiment on their own with tools before feeling comfortable about the tasks. Some examples of certain tools may be provided, such as `telnet` within the Firewall lab. However, the tasks may require a more complex example in order to thoroughly understand how this tool is used in the context of the lab.

Few checkpoints are provided to students, causing students to feel unsure with the direction of their work. Students will need to assume that their work is correct until an error occurs. Unfortunately, few checkpoints may lead students to learning **bad practice**, as they may use a function which should be avoided, but wasn't checked before implementing their solutions. If students feel overwhelmed with the lab material, they may resort to online solutions, which short-circuit the learning process, and where outdated functions or poor solutions are taught to them, rather than what's taught in the lab.

No feedback is provided to students unless they reach out to a teaching assistant (TA) or the instructor. While this approach is typical in many educational settings—since instructors are not expected to guide students through every step of an assignment—it highlights a specific issue in the context of this study: the lack of intermediate checkpoints. Students who frequently seek help may feel lost or uncertain about their progress and may desire more accessible, timely feedback to guide them through the lab tasks.

These challenges are all factors that lead to poor education quality, and may lead to cheating, using AI, and shared solutions among other students. All of these issues will be addressed in this work, either through our new lab template, which includes several

new features to aid students' learning, or through the design of SPHERE, the testbed that supercedes DeterLab.

2.4 Security and Privacy Heterogeneous Environment for Reproducible Experimentation (SPHERE)

In this topic, we will describe the new testbed, SPHERE, which will be used to host our new lab material. The background of SPHERE, its architecture, and new learning environment will be described thoroughly before the new lab design is explained.

2.4.1 SPHERE Definition

As this section implies, SPHERE stands for Security and Privacy Heterogeneous Environment for Reproducible Experimentation. SPHERE aims to transform cybersecurity and privacy research by enabling representative, sophisticated, and reproducible experimentation. This allows researchers to build on the work of their peers, fostering collaboration and accelerating scientific progress. [7] SPHERE is a four-year award funded by the National Science Foundation (NSF), spanning from October 1, 2023, to September 30, 2027 (estimated). The project is led by Principal Investigator (PI) Jelena Mirkovic from the University of Southern California's Information Sciences Institute (USC-ISI). Additionally, the Co-Principal Investigators (Co-PIs) include David Choffnes and Erik Kline with Brian Kocoloski as a former Co-PI. [8]

SPHERE aims to be a testbed with a diverse set of features designed to support cutting-edge cybersecurity and privacy research. The primary goal of this testbed is to advance the nation's scientific discovery in these fields by offering a controlled yet flexible environment for experimentation. Researchers will be able to deploy and evaluate security mechanisms, analyze emerging threats, and test novel privacy-preserving techniques in realistic yet reproducible conditions. Additionally, SPHERE is designed to integrate real-world data sources, heterogeneous computing environments, and scalable infrastructure to support a broad spectrum of experimental needs. By lowering barriers to entry and facilitating collaboration, SPHERE seeks to empower researchers from academia, industry, and government, ultimately driving innovation and enhancing the resilience of cybersecurity and privacy technologies.

2.4.2 Architecture

SPHERE offers a sophisticated architecture for its users. In the following section, we describe its basic specifications and discuss the educational layout of the labs. Curious readers can explore the full architecture map on https://sphere-project.net, where they can view the complete project layout. In this study, we focus specifically on the experiment topologies used in our labs.

Students and teachers are given access to SPHERE's EDU portal, used to create student accounts and distribute learning materials across each of them. These labs are performed within controlled testing environments, and students will not be able to access other users' resources. This server is linked to the USC-ISI's Merge Portal, where several small portals, like Artifact Evaluation Committee, Human Study, and Graphical User Interface portals, are hosted from. There are three testbed facilities, which are located at USC-ISI, a USC colocation space, and at Northeastern University. [9]

SPHERE uses some unique terminology; it will be helpful to cover those definitions before delving into the rest of this section. Some terms from this list include old and new vocabulary. When DeterLab transitioned to SPHERE, it briefly operated under the name "MergeTB", utilizing a network testbed architecture developed by the University of Southern California's Information Sciences Institute (USC-ISI) [10]. SPHERE retains the core design principles of this architecture but adopts new terminology to distinguish itself from MergeTB. References to the earlier vocabulary remain relevant, as SPHERE's command-line interface (CLI) continues to employ much of MergeTB's original terminology.

- **Users** are accounts provided to students, which are recycled at the end of each semester at UMD.
- **Projects** are a container of experiments that are hosted on each account.
- **Experiments** are a set of nodes that are linked together through a topology. A topology describes the layout of connections between different nodes that students will use for their labs.
- **Reservations (formerly called realizations)** are requests for resources, which submit the experiment to SPHERE to "reserve" a collection of cores/RAM for

their topology.

- **Activations (formerly called materializations)** are reservations that have been initiated, which can be attached to an XDC. These are freshly created nodes that will contain students' resources until the activation is deactivated or expired (one week after creation).
- **XDC** is short for an eXperimental Development Container. This is a container (in the sense of a lightweight virtual machine) that activations are connected to, which route the resources from the activation to the user. XDCs are accessed via the web interface, or through SSH if they wish to work within a text-based environment, despite the newly-created labs requiring a GUI to complete.
- **Nodes** are Debian-based systems on which students conduct their work. These nodes function as temporary virtual machines (VMs) that instructors can configure to create a secure "sandbox" environment for students. Some experiments include multiple nodes, which are networked to allow controlled interaction between them.

When experiments are started by the users, a model is constructed by the testbed to prepare a topology of nodes. This is done in Python, and an example is provided below.

```
# Import the Merge library.
from mergexp import *

# Creating a network for the lab.
net = Network('xss', addressing==ipv4, routing==static)

# Defining the nodes.
server,client,teshwan = [net.node(name, image == 'bullseye-edu') for name
↪  in [ 'server', 'client', 'teshwan' ]]

# Create a link connecting the three nodes.
link = net.connect([server,client,teshwan])

# Making the experiment runnable.
```

```
14 experiment(net)
```

This topology is constructed, pushed to an experiment as a revision, and then the experiment is compiled. If there are no errors, it generates the topology that was modeled in the Python. This is an example of the XSS lab's topology (Figure 2.6), viewed on SPHERE's web interface.

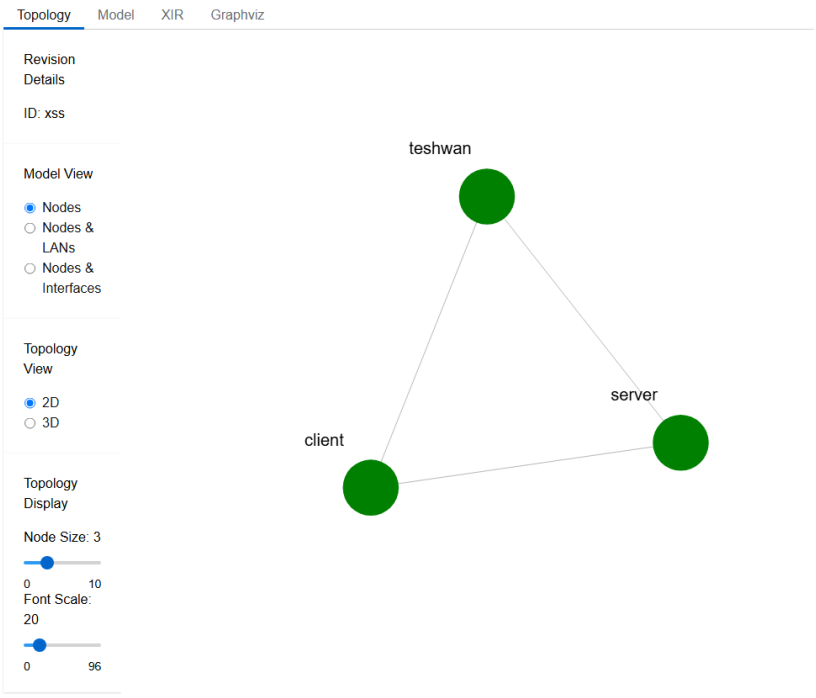

Figure 2.6: Topology Example

2.4.3 JupyterLab Environment

An XDC has been defined in the previous section, defining it as a container that students can access through either a GUI or a terminal. To improve the usability of our labs, we opted for a design that uses the GUI. SPHERE uses a JupyterLab environment, which provides many features that will be described in the next section. However, in this section, we will describe a few basic features of JupyterLab before diving more into the features.

When students navigate to their XDC, they are greeted with an interface that appears in Figure 2.7.

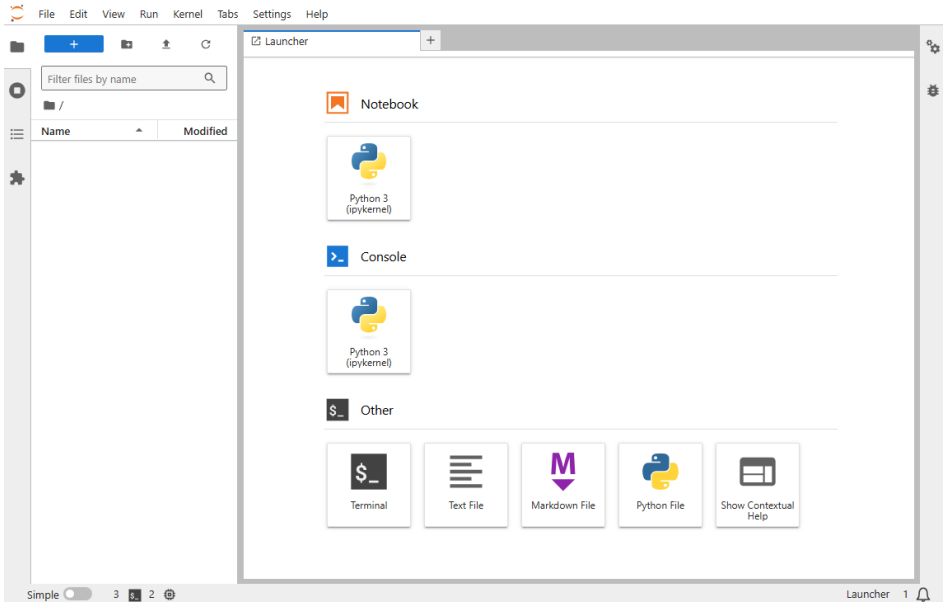

Figure 2.7: JupyterLab Environment

JupyterLab is a non-profit, open-source project that became a separate release from its predecessor, IPython (Interactive Python) [11]. While IPython was initially released in 2014, JupyterLab was officially launched on February 20, 2018, to serve as the next-generation web-based interface for Project Jupyter [12].

JupyterLab is designed to offer a flexible and interactive environment, allowing users to seamlessly engage with their data and code. It supports a variety of workflows, whether for research, project development, or learning. Its highly customizable interface enables users to arrange and organize their workspace according to their specific needs. As a result, JupyterLab serves as an ideal platform for educational settings such as SPHERE, where students require an intuitive and efficient interface to interact with their XDCs.

The layout and tools within JupyterLab are designed to enhance both coding and data exploration. As mentioned earlier, JupyterLab requires only a web browser for access. Its web-based interface eliminates the need for complex installations or local

configurations, allowing students to focus on their tasks rather than troubleshooting software setup. In our redesigned labs, we leverage two core components of JupyterLab:

- Interactive Python notebooks (Jupyter Notebooks) – which provide a structured format for writing and executing code, creating visualizations, and adding explanatory text.
- A built-in terminal - allowing users to execute shell commands and run scripts directly within the environment. The integration of both these features within a single platform fosters innovation in lab design, enabling us to tackle the challenges outlined in Section 2.2.2.

JupyterLab natively supports over 100 programming languages, with Python being the primary choice for SPHERE. [13] Python's simplicity and versatility make it particularly well-suited for driving innovation in our labs. As we explore the specifics of the JupyterLab environment in the following section, it is essential to recognize that its design philosophy emphasizes usability, interactivity, and accessibility, making it an excellent fit for educational and collaborative settings like those found in SPHERE.

2.4.4 SPHERE and JupyterLab Features

Before discussing the innovations introduced in our new labs, it is important to highlight key features of SPHERE and JupyterLab. These features serve as the foundation for the new tools and methodologies that will enhance students' learning experiences in cybersecurity.

SPHERE provides a command-line interface (CLI) that facilitates communication between the terminal and the architecture. The CLI serves as a complete alternative to the web interface, enabling users to perform various actions such as creating accounts, fully configuring experiments, and transferring files. While this CLI does not need to be used by students, it is a critical part of how the new lab design automates tasks. Below is a minimal example of how a lab can be initialized in SPHERE, assuming a project and XDC have already been created:

```
mrg config set server grpc.sphere-testbed.net
mrg login [username] -p [password]
```

```
mrg new experiment [experiment].[project]
mrg realize [realization].[experiment].[project]
mrg materialize [materialization]
mrg xdc attach [xdc] [materialization]
```

As mentioned earlier, the CLI still employs legacy terminology that differs from the updated terms used in this document. Specifically, the verb "realize" was used in place of "reserving", and "materialize" was used instead of "activating". Although SPHERE may eventually choose to update the CLI to align with the new terminology, such a change is unlikely given the system's complexity and the extensive documentation that would need to be revised.

The CLI must be installed on a local machine before it can be used. However, the JupyterLab environment comes pre-installed with the CLI, ensuring seamless access. While students are not required to learn the CLI for their assignments, SPHERE provides pre-written scripts that automate the process of starting and stopping experiments. There are three essential scripts that users use to manage their experiments:

- `startexp` automates the initialization process, providing users with real-time feedback on activation status.
- `stopexp` simplifies experiment termination by relinquishing (deleting) the realization. When a realization is deleted, its activation is automatically removed and detached from the XDC.
- `runlab` executes a pre-written script by the lab's author. Each lab requires an `install` script for each node of the experiment. The `runlab` script calls each `install` script independently, then finishes when all node installations are complete.

These scripts significantly improve accessibility for students, allowing them to configure their labs without requiring extensive knowledge of the CLI. Although the CLI includes comprehensive documentation, it is in the best interest of instructors to prioritize teaching lab content rather than focusing on experiment configuration. While SPHERE's web UI allows users to create experiments, reservations, and activations, the

CLI consolidates these actions into a single command that can be executed as follows: `bash /share/startexp [name]`

Now, we examine the features of JupyterLab. Each environment includes a launcher, giving students the option to open a terminal, Jupyter Notebook, or Jupyter shell. JupyterLab functions similarly to a web browser, allowing users to open multiple tabs with different files or terminals simultaneously. Additionally, windows can be split to create a side-by-side view of two environments, as shown in Figure 2.8.

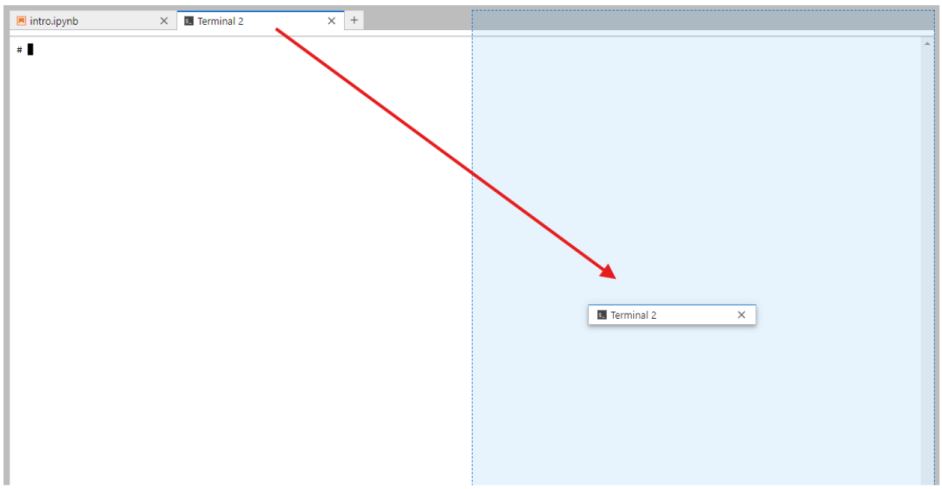

Figure 2.8: Split-Screen Functionality of JupyterLab

Another feature of JupyterLab is access to a file hierarchy. This appears as a sidebar which allows students to navigate across their Debian environment. SPHERE used to provide full access to the XDC from the sidebar. The sidebar can be seen on the left-hand side of Figure 2.7. The sidebar has additional UI features, where the blue button is used to create a new launcher, as seen in Figure 2.7, create a new directory, upload a file, and refresh the sidebar.

Along the very left-hand side of the JupyterLab environment, more advanced features are provided, such as viewing open kernels/tabs, a table of contents (if a notebook is open), and installing extensions. The extensions are provided by a third-party extension community. Since JupyterLab is open-source, users online are free to develop their own extensions and share them online. One extension which will be used in our labs is called `ipywidgets`, which adds interactive HTML widgets into Jupyter Notebooks.

Some examples of these widgets are sliders, checkboxes, text inputs, tabs, accordions, maps, 2D/3D visualizations, datagrids, and a lot more. [14]

2.4.5 Innovation

As described in the previous section, many additional features were not available in the legacy exercises. By leveraging some of these features, we have developed a new, hands-on learning environment for cybersecurity exercises. In late 2023, we decided to convert our labs from DeterLab to SPHERE. This transition not only introduced quality-of-life improvements to our original labs but also provided us with additional time to develop innovative ideas for these labs.

When DeterLab announced its discontinuation, it presented an opportunity to rework the labs using these new features. The legacy labs faced challenges in effectively teaching cybersecurity concepts, as described previously. SPHERE provides a new learning environment, which motivated us to "modernize" certain aspects of the labs. This incentivized a full lab restructure to better facilitate the teaching of the material.

One key motivator comes from OverTheWire: Bandit, a scaffolding learning approach for teaching security concepts. [15] These exercises are available at https://overthewire.org/, where several different "wargames" with various objectives are hosted. Bandit employs a level system that ranges from Level 0 to Level 34, with each subsequent level accessible only after the previous one is completed. Each level contains a hidden key that must be found to progress to the next level.

This approach has been considered for our legacy labs as we attempted to "scaffold" the exercises by breaking a single ultimate goal into more manageable steps. To achieve this effectively, we adopted components similar to those used in Bandit, and Jupyter Notebooks provide a conducive learning environment for this approach. Consequently, the new labs will be described as "banditized".

Not only will the lab content be updated, but the inherent features of a JupyterLab environment will also enhance the lab setting. For example, students previously had to rely on different terminals with varying functionalities and operating systems. Our new material will transition from being entirely command-line focused to using an interactive web environment, thereby improving accessibility for all students. Additionally, although the legacy labs did not require high computational power, the online nature

of the banditized labs relieves students of concerns about system requirements.

One feature provided by SPHERE is that all main computation is performed on testbed resources, reducing the need for students to have powerful devices (e.g., to run multiple VMs). The shift to a user-friendly web interface further lowers the barrier to entry for personal devices. Students concerned about their devices being too slow for computations can now easily access their labs online. This innovation enhances reliability, ensuring that students can complete their work regardless of the device they use.

2.5 Related Work

Many research projects and studies exist for eLearning resources, specifically for math classes, as they contain similar concepts and features found in this study. The goal of these services is to provide an easy-to-use tool to help make learning feel modern, and improve students' comprehension.

Sharp evaluated the use of Codecademy as an instructional supplement in an introductory Python course and found that, while not all targeted outcomes were achieved, the benefits to student learning generally outweighed drawbacks [16]. Wania gathered end-of-term student perceptions of MyProgrammingLab and BlueJ in an introductory course and reported that a large majority found these tools helpful for developing programming skills [17]. These studies motivate our adoption of browser-accessible, interactive labs (via JupyterLab/SPHERE) and inform our E1 design from Section 2.2, which likewise collects post-lab measures of perceived effectiveness and learning.

Another similar work targets comparative evaluations and integrity features. Comparative studies of learning platforms emphasize the role of automated feedback and algorithmic exercises for both learning and academic integrity. Papp compared outcomes across Pearson MyLab and Cengage WebAssign over multiple terms, highlighting immediate feedback, analytics, and algorithmic items that limit answer-sharing [18]. These design elements relate to our second experiment from Section 2.2, which was our new vs. legacy comparison: Our redesigned labs add formative checkpoints and automated validation scripts intended to provide timely feedback while discouraging shortcut-seeking.

Finally, Dvoroková and Kulhánek look into course modernization via publisher and

platform tools. Outside cybersecurity, course redesigns using commercial platforms report improved logistics and added practice/assessment capabilities (e.g., Pearson tools in a redesigned economics course) [19]. While context differs, these results support our rationale to "modernize" legacy labs with a platform that lowers setup costs and increases structured practice. However, because this study involves a custom-designed system rather than a commercial platform, a direct comparison of setup costs is not applicable.

There are many other resources that provide hands-on training platforms for students that are learning about cybersecurity. SEED Labs [20] and Labtainers [21] provide structured, virtualized lab environments using VMs or Docker, enabling authentic system and network security experimentation. While self-contained and aligned with core security concepts, there could be setup challenges for students unfamiliar with virtualization technologies, like Docker. SecKnitKit [22] and NCyTE [23] also support cybersecurity instruction through lecture-aligned exercises and competitions, but rely on virtual images, as well.

Other platforms emphasize gamification or challenge sequencing to drive engagement. NICE Challenges [24] deliver narrative-driven browser-based labs with automated scoring, while OverTheWire [25] provides progressive, SSH-accessible hacking wargames that scaffold small, skill-focused tasks. CyberCIEGE [26] approaches training through a Windows-based network defense simulation game where students make strategic choices and learn from in-game consequences.

Commercial offerings—including Blue Team Labs, Cloud Labs, and CYBRScore Labs [27]—extend similar concepts through scalable cloud-hosted practice environments but introduce access costs. Meanwhile, testbed-hosted cybersecurity materials from DeterLab [6] have reached tens of thousands of students and support realistic experimentation with networks and malware in controlled conditions. However, DeterLab has transitioned into SPHERE, and provides new features that this study will leverage for a new lab design.

This study is unique among related works, as it provides accessibility for students along with automatic setup of the lab environment and automatic installation, which allocates the lab materials across the experimental nodes provided by SPHERE. Additionally, this study aims to improve instructor resources by simplifying grading and

student support. For example, instructors are able to access each student's XDC environment individually, directly from their own account. This leverages SPHERE's shared XDC feature, meaning instructors do not need to continuously log in and out of student accounts. Furthermore, by utilizing SPHERE's command-line interface (CLI), a grading script has been developed that logs into each student's account, uses the `scp` command to copy their results onto the instructor's machine, and then automatically compiles a spreadsheet displaying each student's percentage of correct responses across each lab.

Because these labs include automatic VM setup and resource allocation, they resolve issues present in the SEED and Labtainer labs. While NICE Challenges provide automated grading for students, they lack instructor-oriented tools for assistance and evaluation. OverTheWire offers a similar scaffolded approach in which lab tasks build upon one another; however, it relies on keys that students can easily share, allowing them to skip ahead at any point in the lab. Although preventing cheating remains a challenge for instructors, our labs use randomized values that deter solution sharing. While students can still share code, they are not able to skip to the end of the lab; they must work incrementally since lab steps are automatically generated for them. While it is difficult to fully eliminate academic dishonesty, it will slow it down, and not make cheating as easy compared to related lab resources.

Chapter 3

Methodology

In this chapter, we will describe our procedures, changes, and data collection techniques to evaluate students' comprehension with our new labs.

3.1 Pedagogy Changes

This section examines how each lab was updated and scaffolded to provide improved functionality for both students and instructors. It also outlines how these new design elements aim to enhance the overall learning experience. For reference and comparison, the original lab designs are described in Section 2.3.

3.1.1 Structure of New Material

In the next section, we will describe each lab and its learning objectives. Five of the seven labs are organized into four topics, which follow this structured learning outcome:

- Introduce the basics.
- Introduce the vulnerability.
- Break the vulnerability.
- Fix the vulnerability.

One lab that does not follow this structure is the Intro to SPHERE lab, as it is designed to teach students basic Unix commands and how to navigate within their JupyterLab environment. The other exception is the Pathname lab. The exploit in this lab follows a straightforward process and is expected to be understood quickly. Since students complete the "Introduction to SPHERE" beforehand—where they learn about directory traversal—the Pathname lab skips those basics and instead begins by introducing the vulnerability in Flask.

We adopted this approach for the new lab material to address one of the challenges outlined in Chapter 2.3.2: the lack of intermediate checkpoints in each lab. Rather than requiring students to take "big steps," which can lead to confusion or errors, we break down the material into "small steps," introducing new concepts incrementally.

The final topic of each lab serves as a "large-scale application" of the exploit being taught. These sections are adapted versions of the original lab material, enhanced with improved quality-of-life features. This new structure provides more context and foundational knowledge, enabling students to fully understand not only the exploit but also its broader implications.

3.1.2 Introduction to SPHERE (13 Steps)

The Mann–Whitney U test showed that two out of our twenty-one questions were statistically significant. While this result falls below our expectations, there are important caveats associated with the Mann–Whitney U test, one of which was previously discussed. Although the test does not require the two populations to be of equal sample size, it performs better when the sample sizes are approximately the same [28].

Another issue with the test, mentioned in Section 4.1.2, is that conducting multiple comparisons between groups increases the likelihood of a Type I error—incorrectly rejecting a null hypothesis when it is actually true. To reduce this risk, we applied the Bonferroni correction, which aims to produce more reliable results from the data. A likely reason for our limited number of significant findings is the small sample size of students who completed the "legacy" labs. This issue was addressed during the study and is discussed further in Section 5.2.

One potential mitigation strategy we considered was to combine similar questions.

By reducing the number of comparisons, we could lessen our dependence on the Bonferroni correction. However, this approach led to a loss of specificity in the data. For example, Question 4 and Question 11 both addressed the written materials provided in the labs. Combining them raised two concerns:

1. Students might feel differently about each question—they could favor one while disliking the other.

2. If the combined question were found to be statistically significant, it would be unclear whether Question 4 or Question 11 (or both) contributed to that significance.

Despite these limitations, we can confidently conclude from our results that students appreciated knowing their exact grades prior to submission and viewed the auto-submission mechanism as a useful new feature. With a larger dataset, some of the results that were marginally significant ("Maybe" significance) may reach stronger statistical support.

The Introduction to SPHERE lab serves as a small version of OverTheWire: Bandit from Section 2.4.5, designed to guide students through basic Unix commands. In Bandit, students must locate a key to progress to the next level. Similarly, the introductory SPHERE lab is "banditized"—its exercises build sequentially on one another.

Although the lab is structured to encourage progression, students are not intentionally prevented from continuing if they encounter difficulties. Each step is presented in a digestible format, and built-in feedback is designed to help students overcome obstacles. The lab does not follow a rigid structure; instead, it introduces essential commands that will be valuable in later exercises or for general Unix use.

Students already familiar with Unix will still encounter new material, as SPHERE-specific terminology—such as nodes, XDC, and other platform concepts—is introduced. The lab also trains students to connect to their experimental nodes within the SPHERE environment.

Students will work through a set of basic instructions and commands, which are built to work on top of one another. Within their `intro` node, the students will create a directory, navigate into the directory, create a basic text file by using the `touch`

command, then navigate back to their home directory and delete the directory that they previously made.

The fourth step is similar to the main task in the legacy Intro lab – looking for a file. In the new Intro lab, a file is hidden on their experimental node, and they're told to use the `find` command to locate the file and delete it. Next, students will learn how to download files from the internet by using `wget` to retrieve a file called `sowpods`, which is a word list used in English Scrabble tournaments. Using this text file, students will practice the `grep` command, where they must find all words that start with the word "camp", describing how to use a caret symbol in a regular expression to indicate a match starting at the beginning of a line. (Regex is not heavily covered, but a very basic description is provided to students.)

Upon completing this step, the notebook will automatically generate a directory inside of their home directory called `Important Data/`, which purposefully contains a space in its name. Students practice moving their output from the regex command into this directory, learning about how to escape spaces within a directory name. Following this, students use the `cp` command to create a copy of their output. Once this step is completed correctly, two random lists of strings (called `list1.txt` and `list2.txt`) are generated for them inside of a subdirectory called `lists/`. Note that these lists are the same, except one random line is changed. This new subdirectory is still within the `Important Data/` directory.

For each student, a single random letter in a random line of `list2.txt` is modified- Students must use the `diff` command to identify the change and redirect the output to a new file named `listdiff.txt`. Recall that `list1.txt` and `list2.txt` were identical until the notebook introduced this minor change, requiring the use of `diff`.

In the next step, students learn how to compress `Important Data/`, which contains the work from the previous exercises. They are instructed to create a tarball—a compressed archive commonly used in Unix systems. Once the tarball is created, the notebook automatically verifies that its contents match those of the original `Important Data/` directory.

Once the tarball is created, students are introduced to the `scp` (secure copy) command, which they will use to transfer data from one computer to another. This is done between their `intro` node and their XDC, where the lab silently extracts their tarball to

make sure that it contains the exact content as what was originally compressed. Once this step passes, students are told to navigate to the `intro` node to continue the rest of the lab.

Once the tarball is correctly copied to the XDC, a new file named `numbers.txt` is generated in the student's home directory. Students then interact with this file to learn about Unix pipes and the `sort` command. They must combine these commands to identify all numbers containing the digit 9 and redirect the results to a new text file.

To discourage guessing or shortcutting, the notebook provides a fill-in-the-blank field where students must enter the exact command specified by the lab instructions (Figure 3.1). Because this task can technically be completed without using a pipe or the `sort` command, the automated check enforces their correct use before allowing progression. The notebook executes each submitted command and verifies that its output matches the expected result. If students accidentally alter or delete their `numbers.txt` file, they can rerun the step without input to automatically restore the file.

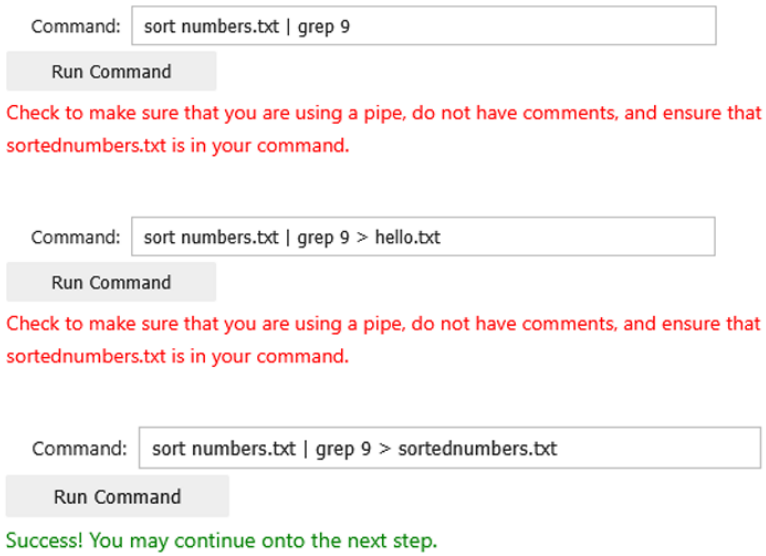

Figure 3.1: Intro Lab Fill-In-The-Blank Example

In the final step, a shell script is automatically generated within their home directory, called `message.sh`. Students are given a very brief description of the `chmod` permission-modifying command so that they can add execution permissions. (The `chmod`

command is covered more thoroughly in the POSIX lab.) Once the student makes the file executable, and the notebook detects this change, the lab is complete. Students may execute the file, if they wish. This script prints: "Congrats! You completed the lab!"

The banditized Intro lab is much more comprehensive than the legacy Intro lab, as it guides students through how a basic Unix environment works. While students work through their lab, they become comfortable working in a notebook/terminal environment. The lab also attempts to teach more knowledgable Unix users about lesser-used commands, such as the (|) operator, and essential commands, such as `scp`, a command that's required in many real-world circumstances.

3.1.3 POSIX Permissions (23 Steps)

The POSIX Permissions lab is the second lab that's offered to cybersecurity students at UMD. This is the first lab which contains a topic breakdown to help guide students from small tasks to a full application. More information about the structure of the new lab material is described in Section 3.1.1.

Students begin learning about symbolic POSIX mode, which is where the Intro lab ended. Students were told in the Intro lab about the `chmod` command, but its syntax and mechanism were not thoroughly described. The lab starts with a breakdown of the structure of POSIX Access Control Lists (ACLs). This teaches students about the three basic access classes and three different permissions for each class. That is, `owner`, `group`, and `other`, which can all have `read`, `write`, and `execute` permissions.

Students are automatically provided with a directory named `posix_practice/`, where three files named `q1.txt`, `q2.txt`, and `q3.txt` have zero permissions, and students must practice a sample set of permissions in the first three questions. This is done by being told a specific set of permissions, then using the `chmod` command to apply them. Students are also provided a text field, where they must answer what the executable bit on directories does. This is to help encourage students to ask questions and re-read the introduction to the lab, where the answer was provided in a short introduction to the lab.

In the next section, students are introduced to octal POSIX permissions notation, a more efficient way to set permissions on files and directories. The drawbacks to symbolic notation are described, and the octal notation is compared to it. Once octal notation

is described, the lab takes advantage of Jupyter Widgets, providing a calculator where students may play around with octal notation (Figure 3.2). Jupyter Widgets is an extension for JupyterLab, which implements all the buttons, text fields, and HTML elements within the labs. This is a common extension for education-based material. [29] The "checkbox" feature from Jupyter Widgets is used for this calculator.

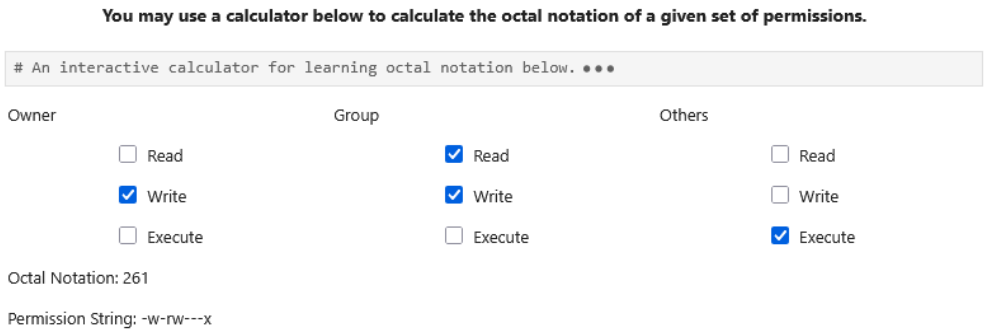

Figure 3.2: Octal Permission Calculator Example

Once this calculator is introduced, students are asked to provide the octal notation of the permissions applied in the first three steps. These are fill-in-the-blank questions, where the notebook compares a three-digit answer from the student to the current set of permissions on the files in the node.

After symbolic and octal notation are introduced, students are given supplementary reading on the three special permissions: `SUID`, `SGID`, and the "sticky bit." The material explains what each permission does, how it is applied, how it appears in a file listing, and how it interacts with octal notation.

Before applying these permissions themselves, students are asked to search the root (`/`) directory to find a certain directory that contains a sticky bit. (The root file system of virtually all Unix systems has a well-known use of the sticky bit.) This is to encourage students to think about the Unix environment setup, and why a sticky bit is applied to that specific directory. This task encourages students to reflect on the Unix environment's structure and to consider why the sticky bit is applied to that particular directory. After identifying the directory and responding to the follow-up question, students practice using octal notation again—this time incorporating special permissions.

Within the final topic of the lab, students are expected to read a scenario and use

their intuition for applying permissions. Students are introduced to two commands used to create users and groups, which are `adduser` and `groupadd`. The notebook does not auto-generate users and groups, giving students the experience with creating and managing accounts in a Unix environment.

Students create four user accounts named after Pokémon characters and form a new group. They then add three of the users to this group, intentionally leaving one out so that the remaining user belongs to the `other` access class. Next, students will create a directory called `/collections/`, within which several files are automatically generated as they progress through the lab. Each new file presents a different scenario requiring students to determine and apply the most appropriate permission settings. In addition to applying permissions, students also learn how to remove them. In the final step of the lab, they practice revoking specific permissions to protect and preserve the directory.

3.1.4 Buffer Overflow (21 Steps)

The Buffer Overflow lab is more complex than the previous two exercises, as it introduces a technically dense topic that requires additional background. This lab covers the first exploit that cybersecurity students encounter in the course and is taught—by necessity—using the C programming language, which many students find challenging. Students learn about a process's address space, including the architecture underlying program execution, and explore the vulnerabilities that mid-level languages such as C can introduce.

Students start off with learning the basics of C, which is the language chosen to demonstrate buffer overflows for the lab. Students are treated as if they do not know how C works; students with a background in C can treat this as review so that they are familiar with the syntax before exploits are demonstrated. Students are provided an initial template of how a very basic C program appears, by explaining the `main()` function and using `<stdio.h>` for input and output. The `printf()` command is provided, and students create a basic program that prints their username. The second step demonstrates how `gcc` is used to compile C programs, and the third step tells students how to run the program. This step is completed automatically when Step 2 is complete, since it just instructs students how to run their compiled binary file, which already has executable permissions by default.

Basic programming syntax is described, such as assigning variables and strings. Students are introduced to the character array in C, which allows a non-null terminated string to be introduced. This is the first function with "unsafe" behavior of C that's showcased, demonstrating what happens when a null terminated and non-null terminated character arrays are assigned near each other in the stack. Figure 3.3 is provided to students to provide a demo of this exploit in C, showing the dangers of writing poor C code.

```
char good_string[] = {'h', 'e', 'l', 'l', 'o', '\0'};
char bad_string[] = {'w', 'o', 'r', 'l', 'd'};
printf("%s\n", bad_string);

>> worldhello
```

Figure 3.3: Stack Exploit Example

After strings, pointers are introduced to students to demonstrate the power of middle-level languages. This teaches them how addresses are manually handled in C, and provide students with hands-on scenarios where an address must be handled when sending stored variables to functions. Students that are used to easier programming languages, such as Python, are shown pointers so that they are not confused about the syntax in the final section of the lab.

The second portion of the lab introduces students to unsafe string functions in C. This presents the idea of "unbounded functions"—functions that do not check the bounds of the memory they write to. These functions are `strcpy`, `strcmp`, `strcat`, and `sprintf`. Examples of these functions are provided to students, with each subsequent example file automatically generated as they complete the previous one. Students are expected only to modify the payloads within these examples, focusing on exploiting the functions rather than configuring the environment to make the examples run.

The third portion of the lab introduces students to the safe alternatives of the previously discussed functions. The notebook automatically copies the broken versions that produced segmentation faults, and students are required to replace only the unsafe function they modified, not the payload. The notebook verifies this by comparing each student's broken and fixed versions to ensure that the payload remains unchanged and

that the correction results solely from substituting the unsafe function. Safe alternatives, along with their function definitions and official documentation, are provided for students to review and implement.

In the final topic of the lab, the students are provided a full application that's littered with unsafe C code. To help emphasize the dangerous nature of this exploit, the application is called Wormwood, a nuclear reactor simulator written in C with an text-based interface provided by the `ncurses` library. This library creates a more robust user-interface, providing a more lively environment that students will be cracking (Figure 3.4). The students are provided four different vulnerabilities that are hidden within the program. Some of them were covered previously, but some of them are also basic programming errors demonstrating the lack of safety checks within C.

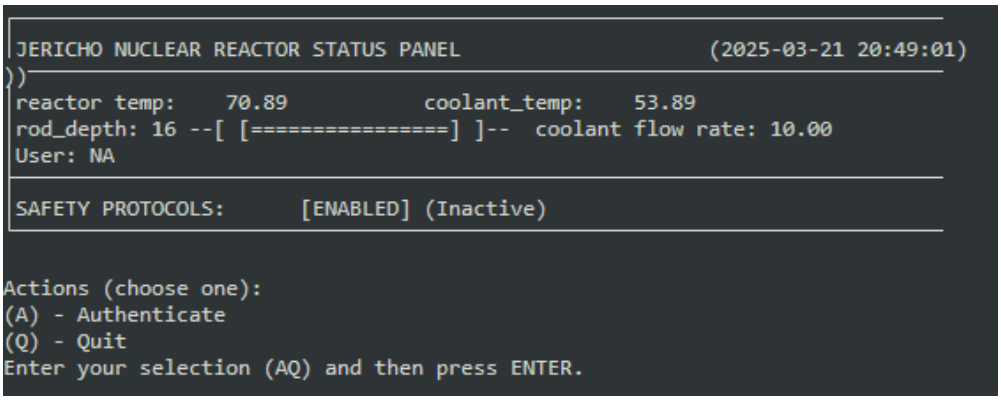

Figure 3.4: Wormwood GUI

Students are introduced to four different vulnerabilities in the program, all of which still allow the C code to compile successfully. These vulnerabilities include a buffer overflow, an off-by-one error, an integer overflow/underflow, and a format-string vulnerability. Each step requires students to exploit one of these vulnerabilities individually by providing input that causes the program to fail.

For two of these exploits, students complete three fill-in-the-blank fields: the password prompt, the menu option, and the payload. By this point, it is assumed that students have already completed the format-string vulnerability, which reveals the password of one of the users. Students then use this password, select the menu option they believe is vulnerable, and enter the corresponding payload. The notebook verifies that

distinct payloads are used for the off-by-one and integer overflow/underflow exploits, and it also detects the specific location where the student attempts the exploit. Before executing the payload, the notebook checks the selected menu option and provides corrective feedback if an incorrect option is chosen.

When a student submits a payload, the notebook launches a process to test the input using a Python script stored on the experimental node. The script employs the `subprocess.Popen` command to send the input, flushing the `stdin` buffer after each entry. A brief pause of half a second occurs between inputs, after which the script issues a `communicate` command to capture and evaluate the output. The step is marked complete when the output matches the expected result.

After testing all four vulnerabilities, students proceed to another instance of the program, Wormwood, where they must apply patches to correct each issue. In this phase, students maintain both a "test" and a "fix" version of Wormwood. The notebook reuses their previous exploit inputs, first executing them against the unpatched version to confirm that the vulnerabilities originally worked. If these payloads successfully break the unpatched program, they are then executed against the patched version. The notebook evaluates the outputs, and if the patched program behaves correctly without errors, the corresponding step is marked as passed. The lab concludes once all four vulnerabilities have been successfully patched.

3.1.5 Pathname Attacks (12 Steps)

The Pathname lab teaches students about directory traversal attacks within the address bar of a browser. This allows students to view files on the web server that are not supposed to be accessed on the front-end of a website. Despite this lab being shorter, it requires port forwarding to be set up so that students can access the web server from their personal computers. To make port forwarding easier, it is set up using a separate notebook that is provided to students. The port forwarding process will be explained in Section 3.1.2, where additional features of these labs will be discussed. Once port forwarding is configured on the student's machine, the lab may begin.

The first step is ungraded, as it instructs students to set up port forwarding on their system. Due to the importance of configuring this, it's treated as a step in the notebook since it takes a while to tunnel the node's resources to the machine. A visualization

of port forwarding is described as an optional reading for curious students. The setup requires a different `SSH` command to be used in order to properly connect to the node, and for the resources to be available in the browser.

The second step begins to teach the student about the lab material, as a Python library called Flask is introduced to the student. Flask is a framework used for prototyping a web server, which is suitable for the needs of this project. Since Python's difficulty is comparably easier than C, it has less of an introduction compared to the Buffer Overflow lab. However, templates are still provided to the student so that there is less focus in debugging, and more focus on learning about pathname attacks and how to remediate them. In this step, students are given a template of a basic Flask application that prints a single word wrapped in a `<p>` element in HTML. Then, a command is provided to start the server. The step checks to make sure that the Flask application runs properly, and the web page is accessible.

Once a basic Flask demo is presented, the idea of a "template" is described to students, which are used to print more complex web pages. A template is pre-written for the student, and the student must follow the directions by using a command called `render_template()` in order to display it within the Flask application.

After the basics of Flask have been demonstrated, this lab begins something unique which is not seen across other labs. The student will be in a pair-programming setting, where a fictional "developer" (portrayed by the notebook) is working on a web application with the student. The developer is implementing the heavy coding for a memo website, and the student must complete some of the simpler tasks so that they have a better understanding of Flask's functionality. The developer is introduced to the student, where the developer creates a directory called `/lab` which contains code that they have already implemented. The student must implement a redirect in Flask, where they are provided the `url_for()` command, and they must redirect the user after `add_memo()` and `delete_memo()` have been called. Students are given a `cURL` command to assist them with debugging their function, as the functionality has not yet been implemented onto the HTML page. This provides them with some basic back-end knowledge and helps them learn not to rely on debugging from a user interface.

After students complete the previous task, the developer implements the `add_memo()`, `view_memo()`, and `load_memos()` functions for the website. The student is provided a

template to implement `delete_memo()`, which is the easiest function to implement, as they just need to delete a file in Python by using the `os.remove()` function. Once this step is complete, the website is implemented, and the student moves onto a completely different topic, which discusses path canonicalization.

This next topic described how files are read from the address bar into the Unix environment. The `(..)` operator is described to students, which teaches them how to move backwards in a file hierarchy. Once students discover this vulnerability, they will be given some example sanitization functions, which are not provided as code. Instead, they are being told what the mitigation is, which they must attempt to bypass. Hints are provided to the students, and students must provide answer to fill-in-the-blank responses, where a sample URL is prepended to their answer to guide them easier (Figure 3.5).

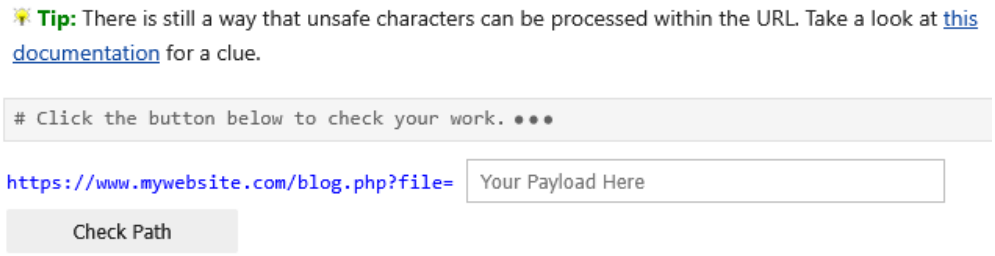

Figure 3.5: Pathname Attack Sample Question

By breaking these incomplete sanitization functions, we hope to convince students not to write their own sanitization functions, but use built-in ones instead. From here, the students will return back to their Flask application that they constructed with the developer. An "oversight" made by the developer is that the application does not canonicalize pathnames when viewing memos. (This is another reason why the `view_memo()` function was not written by the student—it makes students practice fixing code that they did not written.) Before the vulnerability is fixed, the notebook suggests a way to type in a payload to leak a file called `/etc/passwd` onto the website. This payload is tested, then the notebook displays the HTML response of their payload. Figure 3.6 shows the power of the notebook, guiding the student to the correct answer with hints tailored to their response.

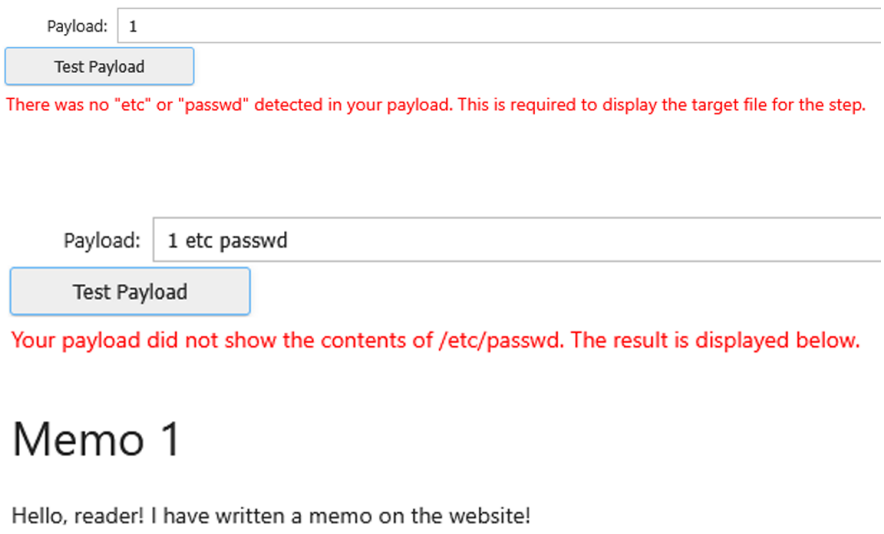

Figure 3.6: HTML Response Example in Pathname Lab

In the final part of this lab, the notebook injects a section of code where the student must provide a sanitization function that uses a built-in "canonicalize" feature in Python. Students are provided some functions in Python, and are asked to check the pathname of the memo before it's displayed to the user. The notebook strongly indicates that any code placed outside of the code block will cause their step to fail, which locks the student out from tampering with pre-written code. This is so that students can't skip the tested task by removing part of the program.

3.1.6 SQL Injection (20 Steps)

The SQL Injection (SQLi) lab teaches students about the risks of making database calls without validating and sanitizing user input. Much like the pathname lab, this lab requires port forwarding. This lab assumes that students have no prior SQL knowledge, and the first topic introduces them to creating a database, table, insertions, selections, and deletions. The database that students create is a small student directory containing an ID, name, and grade. This just covers the minimum SQL knowledge required to complete the lab. An additional "intermediate" SQL exercise is provided but left ungraded. This exercise introduces optional clauses that students may use in the final topic of the lab, such as `LIMIT`, `OFFSET`, `ORDER BY`, and `GROUP BY`.

Once students understand basic SQL queries, the notebook introduces PHP, an HTML-embedded scripting language that can dynamically generate basic webpages with SQL calls. The PHP syntax is not formally introduced; however, students are provided with a basic PHP file that includes extensive comments to guide them in understanding how PHP can be used to execute SQL statements. This topic uses the database that students created in the previous topic, so they are already familiar with the schema before modifying its contents.

The provided PHP file prompts for a student ID. When students input a value, the page displays the table if the query returns any results. Additionally, a helpful debugging message is provided — the exact SQL query used to display the data is shown to students. An example is shown in Figure 3.7.

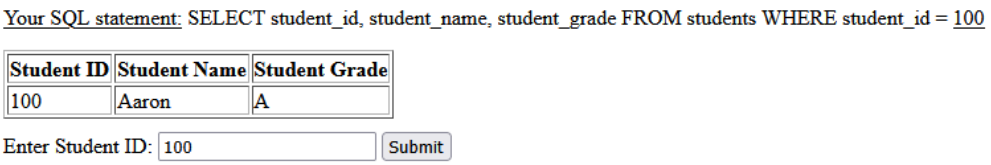

Figure 3.7: Database Example in SQLi Lab

Once students become familiar with SQL calls in PHP and understand how SQL injection occurs, the next topic walks them through the process of creating a prepared statement. These statements are the best mitigation for SQL injection, and a detailed to-do list of changes is provided. Students follow this list to patch the vulnerability step by step. Because several changes are required, constructing a single prepared statement becomes its own topic. This approach allows students to take their time and carefully construct a patch before applying it more broadly.

In the final topic, students are directed to a simulated bank website, where they must use everything they have learned to access user accounts and transfer funds. Within the website's source code, unsafe SQL statements are surrounded by comments containing parameters that students must patch. An example of how to apply these changes is provided, giving students a copy-paste template to assist them (Figure 3.8). The notebook iterates through all nine statements that must be patched and informs students of any missed issues in each block, ensuring thorough coverage.

To assist you with fixing these statements, every query that needs to be updated in `FCCU.php` will appear like this:

```
/*** Block #1: CONVERT TO PREPARED STATEMENT ***/
$query = "SQL statement here";
$result = $mysqli->query($query) or die($mysqli->error());
$row = $result->fetch_array();
/*********************************************/
```

Here's a template of a fixed statement:

```
/*** Block #1: CONVERT TO PREPARED STATEMENT ***/
$query = "Updated SQL statement here";
$stmt = $mysqli->prepare($query);
$stmt->bind_param("si", $string_param, $int_param);
$stmt->execute();
$result = $stmt->get_result() or die($mysqli->error());
$row = $result->fetch_row();
/*********************************************/
```

Figure 3.8: Provided Prepared Statement to Students

3.1.7 Cross-Site Scripting (12 Steps)

The Cross-Site Scripting (XSS) lab demonstrates the dangers of unsanitized JavaScript input. The lab covers two vulnerability types via both `GET` and `POST` requests. The first exercise presents a basic injection attack on a notes website that does not sanitize user input. Authentication for signing into accounts is handled via the address bar (an insecure mechanism that nevertheless helps students understand how the attack operates). Students enter JavaScript into the site, which prints output to their terminal—an elementary demonstration of how injection works. Next, they are given a `fetch` statement that writes to another user's memo page. Finally, students receive a copy-and-paste command (explained in the lab): a nested `fetch` that retrieves another user's authentication token and allows them to sign in as that user.

The next section introduces PHP sessions and the `document.cookie` variable. This background leads into the large, hands-on exercise, in which students access a web forum called "Sloths Unlimited" and are instructed to exfiltrate an administrator's authentication cookie. Authenticating as the administrator allows students to create a category on the site (Figure 3.9), an action unavailable to standard users. To achieve this, students

craft an attack that causes the administrator to execute a script named `steal.php`, which captures the admin's cookie and—using the resulting elevated privileges—creates the new category. The implementation details of `steal.php` are omitted, as they fall outside the lab's scope. The student breaks three hand-written sanitization functions, with a fourth one being optional for curious students.

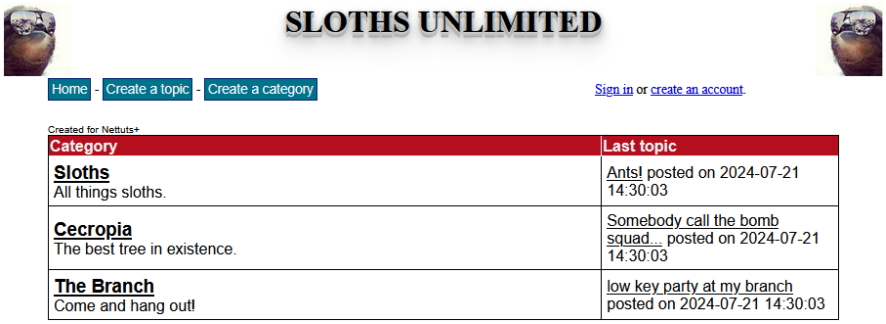

Figure 3.9: Sloths Unlimited Website

At the end of the lab, students are introduced to the `htmlspecialchars()` function, which is used for input sanitization in PHP. Students apply this patch within the `sanitize.php` file, which contains different, hand-crafted sanitization functions that they must break using critical thinking. These functions attempt to remove `<script>` tags through various methods, and students must find ways to mitigate or bypass them.

3.1.8 Firewalls (18 Steps)

The final lab that students complete is the firewall lab, which teaches them about stateful firewalls. The first topic introduces various networking commands that assist in testing firewall rules. The second topic introduces `iptables`, providing students with commands to add, test, and delete firewall rules. This teaches the fundamentals of how `iptables` functions.

In the third topic, students learn the components of an `iptables` rule by constructing a complex rule piece by piece. Each parameter is introduced one at a time. Students use previously learned networking commands to test their rules and identify required attributes, such as the correct network interface. A table is provided during each step,

highlighting key letters in bold to help students remember what each parameter represents (Figure 3.10). At the end of the topic, students are shown the final rule with a full breakdown of its functionality.

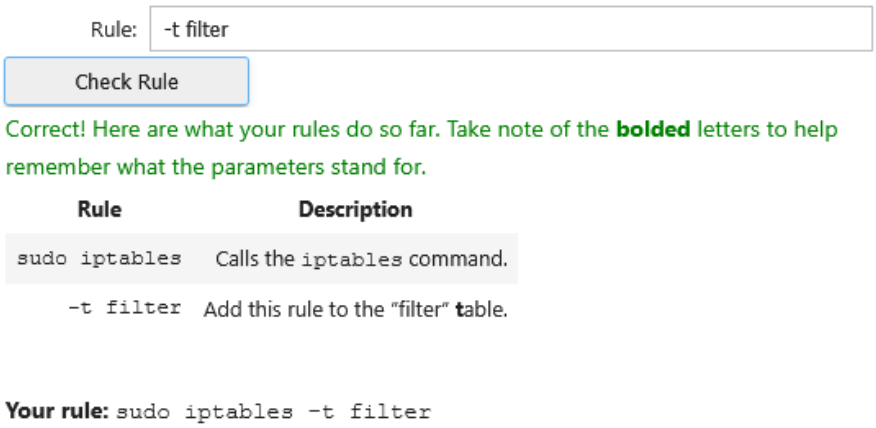

Figure 3.10: `iptables` Table Example

In the final topic, students are given five scenarios in which they must apply the rules learned in Topic 3. To assist them, example console outputs are provided to illustrate what a successfully applied rule looks like. The titles of each step describe the connection direction and protocol, such as "Inbound TCP Connections to the Apache Server" and "Outbound TCP OpenSSH Requests." Warnings are provided when a rule could be "dangerous"—i.e., where typos could result in the student being locked out of the system. Mitigations for data loss are detailed in Chapter 3.1.9, and relevant recovery commands are pointed out to highlight the importance of careful rule application. Teaching firewall rules within SPHERE's controlled testing environment helps prevent costly mistakes students might make in real-world deployments.

3.1.9 Lab Features and Targeted Solutions

By converting these labs into Jupyter Notebooks, we define the term "Jupyterize" to describe the transformation involved in adapting traditional labs into modern, interactive formats. By Jupyterizing our lab material, we automatically gain new features, as explained in Section 2.4.4. These features are provided by JupyterLab itself and address

several pedagogical challenges discussed in Section 2.3.2:

- The **terminal-only interface** has been eliminated, as students can now work in a graphical user interface (GUI) with browser-like features. This allows students to interact with both the terminal and the Jupyter Notebook simultaneously (Figure 2.8).
- Building on the point above, the **user interface** is significantly improved. Buttons can be implemented directly within the notebook to simplify common tasks, such as starting or stopping the lab environment.
- **Automatic resource allocation** and **loading times** have improved substantially, as the JupyterLab environment is more responsive than Deterlab's custom testing infrastructure.

To address the remaining challenges, we have implemented custom functions that leverage SPHERE's CLI. Some existing SPHERE CLI functions—such as starting, installing, and stopping a lab—are now triggered by interactive buttons. The output of the original scripts is condensed into scrollable text regions to aid in debugging when issues arise. The following features were developed to address challenges outlined in Section 2.3.2:

- An **auto-save** and **auto-load** feature is implemented for each notebook. When a student checks their work at any step, the notebook captures the current state of their experimental node, compresses it into a tarball, and stores it in the student's JupyterLab environment. If the lab expires or needs to be restarted, the student can simply click "Start Lab" and then "Load Lab" to restore their previous session. If the student forgets to load their lab, the notebook will issue a warning. Clicking the button again assumes the student wishes to overwrite the autosave.
- Some labs—such as POSIX—**provide hands-on demonstrations** to better explain key concepts. Examples of these are shown in Figures 3.2 and 3.10, which demonstrate interactive elements designed to help students connect theoretical ideas to lab tasks. These features allow students to treat labs more like a sandbox than a rigid assignment.

- **Checkpoints** and **feedback** are provided at each step. In some cases, students cannot proceed unless the previous step is completed. This reinforces mastery of fundamental concepts before introducing more advanced material. Feedback is dynamically generated based on the current state of the lab and is intended to guide the student toward the correct solution—serving as a virtual mentor before instructor or TA intervention is required.

In addition to solving the challenges outlined in Section 2.3.2, several new features complement the above. These include HTML responses from websites after testing payloads, automatic file generation as students progress, personalized notebooks and commands based on student usernames, exploit diagrams, and loading spinners for time-consuming tasks or delayed step rendering.

3.1.10 Grading

Grading has been entirely re-designed and is explained in detail here. Recall that grading the legacy labs were done completely manually, as mentioned in Section 2.3.1. At the end of each notebook, students are provided with an auto-grader that identifies which steps are correct, incorrect, or optional. When a student wishes to check their performance, they can click the "Calculate Grade" button, which evaluates all completed steps and prints a summary of their lab results (Figure 3.11). This is a newly developed feature specific to our Jupyterized lab structure.

Step 8 is complete.
Step 9 is optional, and does not count towards your final grade.
Step 10 is a reading section.
Step 11 is complete.
Step 12 is complete.
Step 13 is incomplete.
Step 14 is incomplete.
Step 15 is complete.
Step 16 is complete.
Step 17 is incomplete.
Step 18 is complete.
Step 19 is complete.
Step 20 is incomplete.
You have 16 out of 20 steps completed.

Figure 3.11: Grading Within Notebooks

To enable instructors and TAs (hereafter referred to as "graders") to evaluate student submissions, a log is maintained within each student's JupyterLab environment. This log is appended every time work is checked and contains the following information: the student's username, timestamp, lab identifier, notebook response, and user input (if applicable). The log location is hidden from students to prevent tampering.

Graders are provided with a script that requires minimal setup. Full instructions are documented in the "SPHERE instructor" GitHub repository[1] . The grader creates a copy of a script called `grade_students.sh` and replaces placeholder credentials (e.g., `umdclassXXX1:password1`) with actual student usernames and passwords. Additionally, key-value pairs are created to map usernames to real names (e.g., `realName["umdclassXXX1"] = "Austin A"`).

Once configured, the script is run from a Linux environment with access to the SPHERE CLI. This script **cannot** be executed in the grader's JupyterLab environment, due to SPHERE's security restrictions that prevent signing into multiple accounts. After execution, a directory containing all student logs is stored on the grader's machine. A

[1] Access to this GitHub repository is private, but an access request can be made.

summary log is also generated, showing each student's grade as a percentage (Figure 3.12). Once grades are acquired, the instructor or teaching assistant may type them into their course management system.

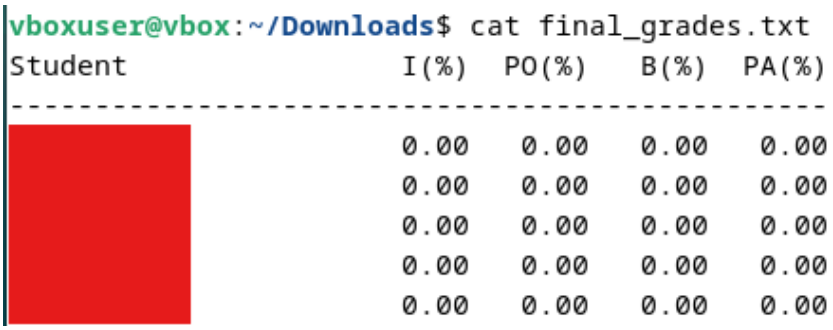

Figure 3.12: Grading Script Output

3.1.11 Templates for Instructors

Instructors who want to create new labs in the style of the ones described here can take advantage of the Jupyterized lab structure by using a script to generate a new lab template. This script automates much of the setup process, simplifying the creation of custom labs for use within SPHERE. Similar to the grader script mentioned previously, documentation for this process is available in the "sphere instructor" GitHub repository. While the script does not fully automate 100% of the lab and notebook creation, it greatly reduces the manual effort required.

To generate a new lab, instructors follow these steps:

1. Download and run the provided `install_notebooks.sh` script.
2. Download and run the `generate_notebook_template.sh` script.
3. Provide the script with the desired sections and steps, along with functionality types (e.g., button, input field, or large text area).

After completing these steps, a template is created that includes pre-implemented buttons which initially return "correct" by default. It is up to the instructor to implement the logic that checks the correct file(s) for each step. The script also generates

the lab topology (see Figure 2.6), the checker files for each button, the notebook, the core control buttons (start, load, and stop), the auto-grader, and the buttons and input fields to check each step.

3.2 Evaluation

To conclude the methodology chapter, we are going to describe how the labs will be evaluated by the students.

3.2.1 Consent and Administration

We evaluated the labs using groups of students in UMD's Computer Security course. Prior to conducting the study, we submitted a Human Research Protection (HRP) review request to the university's Institutional Review Board (IRB). The board reviewed our proposed methodology and determined that the study does not meet the criteria for human subjects research, as defined by federal regulations. As such, IRB approval was not required.

Although formal IRB approval was not necessary, we still follow ethical guidelines in administering the study. Students are informed that participation is voluntary, and their responses are anonymized before analysis. No personally identifiable information is stored or associated with our students' data. Participation or performance has no impact on their course grades.

We first assessed students who did not use the Jupyterized lab material. This cohort used the "Legacy" lab content (described in Section 2.3) because the new lab content did not exist or was not ready for use. This cohort had to manually run scripts to start, install, and stop their labs. These students did not work with Jupyter Notebooks and were given the non-scaffolded material that our updated content aimed to improve. Once the semester concluded, students were invited to complete an anonymous end-of-semester survey. This process was conducted across two semesters: Fall 2023 and Spring 2024, each with a new set of students.

After the development of our Jupyterized labs, they were delivered to a new cohort of students the following academic year. Before this cohort was assessed, students were provided with a consent form and asked to opt in or opt out of the study. During the

semester, both opt-in and opt-out students were given pre- and post-quizzes to assess learning outcomes. At the end of the semester, the same end-of-semester survey from the prior year was administered.

Students who opted out were still required to complete the quizzes and survey to ensure fairness in workload. However, their data was de-anonymized and excluded from research analysis, ensuring that only opt-in responses were retained for evaluation. This phase of the study was conducted across two additional semesters: Fall 2024 and Spring 2025, again with distinct student cohorts.

We assume that the two cohorts are more or less equivalent, and so can be compared as a control and an experimental group. The primary purpose of this comparison was to see if students using the Jupyterized labs had a better experience or impression of the labs than the students using the Legacy labs.

3.2.2 Timeline

- Summer 2023: Legacy labs were ported to SPHERE and prepared for classroom delivery.
- Fall 2023 — Spring 2024: Students were given non-Jupyterized lab material on SPHERE.
- Summer 2024: Jupyterized labs were developed.
- Fall 2024: Jupyterized labs were deployed. HRP approval received. Survey sent to non-Jupyterized participants. Survey and pre-/post-quizzes administered to first Jupyterized cohort.
- Spring 2025: Second Jupyterized cohort received deployed labs and completed data collection. Follow-up survey reminders sent to non-Jupyterized participants.

3.2.3 Pre-/Post-Quizzes and Survey

To measure students' comprehension (i.e., to see if the labs are effective at teaching the material), we administered pre- and post-quizzes to those using the Jupyterized lab material. A pre-/post-quiz was required for each lab, regardless of whether a student opted in or out of the study. As mentioned in the previous section, this ensured a

consistent workload for all students. Students who did not opt in were de-anonymized so that their responses could be excluded from the research data (i.e., students who did not opt in were simply treated as students in the course).

These quizzes were delivered through Canvas and consisted of four open-ended questions related to each lab. Students took the quiz before starting the corresponding Jupyterized lab and—for the purpose of course credit—were graded solely on completion, not correctness. Students were instructed not to use external resources while answering the questions. Canvas includes a feature that allows responses to be anonymized by removing student identifiers, which prevents us from tracing submissions back to specific individuals. As a result, we are unable to measure individual improvement and must instead evaluate learning gains across the cohort as a whole.

We chose to use open-ended questions to reduce the likelihood of students guessing correct answers by chance. Students were encouraged to respond with "I don't know" if they were unsure, and were reminded throughout the study that honest responses were essential for producing accurate research findings. Because quizzes were graded on completion, students were more likely to provide uncertain or incomplete responses when they lacked knowledge, rather than attempting to guess. This also allowed us to maintain anonymity; for example, if five students did not receive full completion credit while 40 did, we could identify only those five students in the gradebook, while the remaining responses remained anonymous.

The end-of-semester survey was administered during the final week of classes, after students had completed the last lab. This timing ensured that all participants had completed the full seven weeks of Jupyterized lab content. The survey consisted of 21 statements, each rated on a 1–10 Likert scale (strongly disagree to strongly agree). A wide scale was provided so that we would receive finer feedback from students, versus a scale from 1-5. The questions targeted areas we aimed to improve through the new lab format. As with the quizzes, the survey was administered to both non-Jupyterized and Jupyterized cohorts, and their responses were compared. The process for analyzing these results is described in the following chapter.

Chapter 4

Results

In this chapter, the results of our study will be provided, and we will introduce the methods used to evaluate our data.

4.1 Usability and Lab Experience

At the end of the semester, students were asked to complete a survey regarding their experiences with the Jupyter-based environment developed for this course. The instrument gathered both quantitative and qualitative feedback, focusing on usability, engagement, instructional support, learning outcomes, and overall confidence with computational tools. As described in the previous section, each question in the survey used a Likert scale with a range of 1–10.

Because of the ordinal nature of the questions, we cannot assume any particular distribution for our results; therefore, we treat the data as non-parametric. Two examples from a single semester are provided below. This does not reflect the full dataset (Figure 4.1).

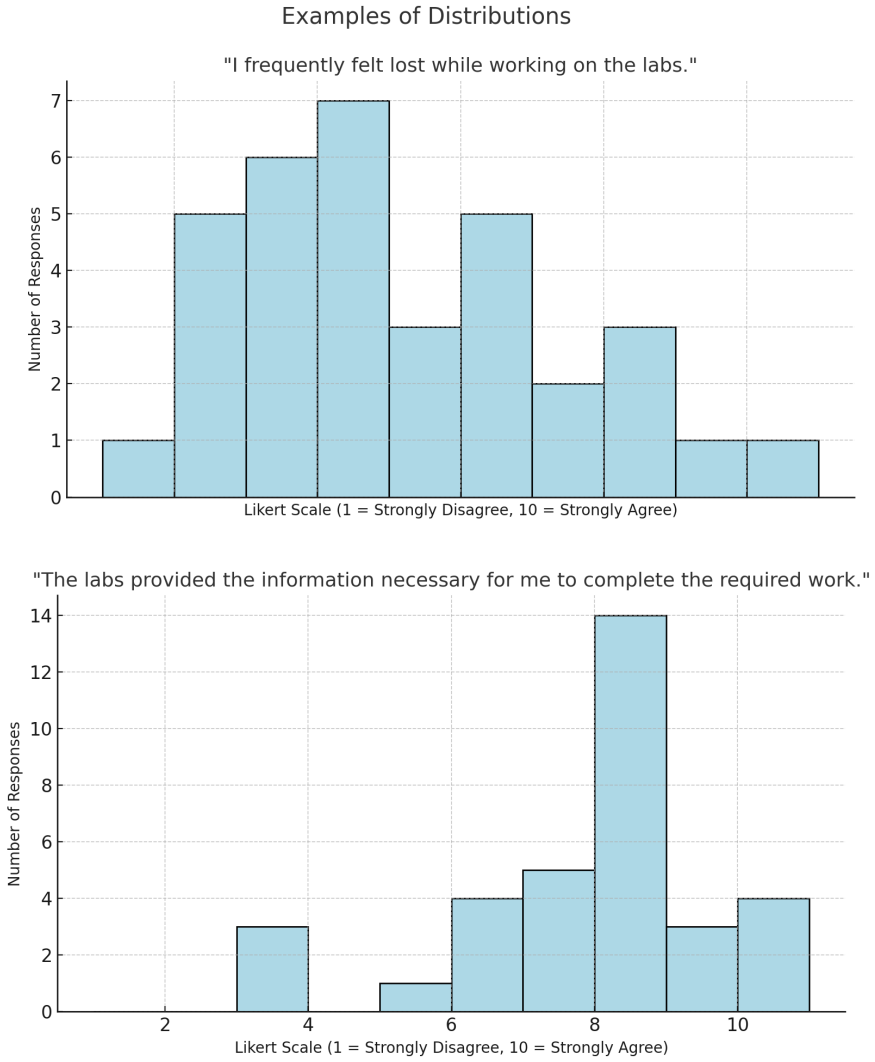

Figure 4.1: Distributions of Likert Scale Results

4.1.1 Hypothesis Testing

All 21 survey questions required non-parametric hypothesis testing. The t-test is a common hypothesis test that's used for testing two, independent groups. However, this requires that the data is normally distributed. Because the assumptions of the t-test could not be satisfied, we used the Mann–Whitney U test instead. This test is suitable for ordinal data and for comparing two independent populations, as in our case—students

surveyed before and after the course redesign. Additionally, the Mann–Whitney U test accommodates unequal sample sizes, which applies here: 19 students provided feedback before the redesign and 72 students afterward. Something to note is that the t-test evaluates the mean, but the Mann-Whitney U test evaluates the median.

The Mann–Whitney U statistic is calculated as follows:

$$U = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

where n_1 and n_2 are the sample sizes of the two groups, and R_1 is the sum of the ranks for group 1.

The smaller of U_1 and U_2 is used as the final test statistic. For large sample sizes, the distribution of U can be approximated by a normal distribution with:

$$z = \frac{U - \mu_U}{\sigma_U} \quad \text{where} \quad \mu_U = \frac{n_1 n_2}{2}, \quad \sigma_U = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}$$

In practice, the SciPy library in Python was used to perform these calculations automatically. The pre- and post-change responses were stored in separate datasets, cleaned, and analyzed using the Mann–Whitney U test.

For each survey question Q_i, a Mann-Whitney U test was conducted to evaluate whether participants' responses differed significantly before and after the intervention. This non-parametric test was chosen because it does not assume normality and is appropriate for ordinal data such as Likert-scale survey responses.

4.1.2 Lab Quality Results

Before applying the test described in Section 4.1.1, we received 19 results from students that took the "legacy" labs. Additionally, we received 61 results from students that took the "banditized" labs. We are going to define questions from the end-of-semester survey as follows (Table 4.1).

Table 4.1: Survey Questions Administered to Students

Q#	Question
1	I found the content of the labs to be engaging.
2	The activities in the labs made me more interested in the subject matter.
3	I frequently needed to ask for help from the TA, professor, or online.
4	The labs provided the information necessary for me to complete the required work.
5	I learned how to do important things through these labs.
6	Each lab helped me understand important concepts relevant to computer security.
7	The labs helped me prepare for the exams.
8	I feel like I had enough time to complete each lab.
9	I learned valuable information from each lab.
10	The lab user interface was user-friendly.
11	The steps I had to take to start each lab were easy.
12	The introductory text of each lab helped prepare me for what was to come.
13	I found the text for each lab easy to understand.
14	It was easy for me to read each lab manual and work on the lab tasks at the same time.
15	It was easy to follow the steps of the labs.
16	I felt that each step in the labs built off the previous one.
17	The tips and warnings that were provided throughout the labs helped me understand what to do.
18	It was easy to save and restore my progress so that I could work on a lab over multiple sessions.
19	It was easy for me to know how well I had done on the lab before submitting my work.
20	The mechanism to get my work to the grader was convenient.
21	I frequently felt lost while working on the labs.

With these questions defined, we will now go ahead and proceed with the Mann-Whitney U Test. We must state the hypotheses before we apply the test:

$\mathbf{H_0}$: There is no difference in results between the "legacy" labs and SPHERE labs.
$\mathbf{H_1}$: There is a difference in results between the "legacy" labs and SPHERE labs.

In this context, rejecting the null hypothesis H_0 indicates that the change between the two conditions (before vs. after) is statistically significant. This suggests that the change

in the lab's feature had an effect on participants' responses for that question. Note that a "feature" is a change between the original and updated labs, which we expect to see improvement on.

The statistical significance of each Mann-Whitney U test was determined using the associated p-value. A significance level of $\alpha = 0.05$ (corresponding to a 95% confidence level) was adopted for all analyses.

The decision rule applied was as follows:

$$\text{Reject } H_0 \text{ if } p < \alpha = 0.05; \quad \text{Fail to reject } H_0 \text{ if } p \geq 0.05.$$

With our null and alternative hypotheses stated, we can look at Table 4.2 to view the results.

Table 4.2: Mann–Whitney U Test Results

Q#	**U-Test**	**p-Value**	**Significant?**
Q1	234.0	0.065275	No
Q2	257.5	0.163816	No
Q3	267.0	0.232788	No
Q4	295.5	0.495314	No
Q5	248.5	0.116712	No
Q6	335.0	0.968880	No
Q7	382.0	0.367976	No
Q8	340.0	0.891015	No
Q9	311.0	0.690774	No
Q10	258.5	0.179018	No
Q11	283.5	0.374127	No
Q12	307.0	0.644957	No
Q13	235.5	0.076254	No
Q14	146.0	0.000585	Yes
Q15	237.5	0.082895	No
Q16	266.5	0.222681	No
Q17	351.0	0.739100	No
Q18	256.0	0.162654	No
Q19	128.0	0.000077	Yes
Q20	118.0	0.000058	Yes
Q21	257.0	0.168930	No

The Mann–Whitney U test evaluates each question independently, which introduces the multiple comparisons problem. When multiple statistical tests are conducted simultaneously, the probability of Type I errors (false positives) increases. While our initial results are limited, we can apply a correction to control for this risk.

To address this, we used the **Bonferroni correction**, which adjusts the significance threshold (α) to account for the number of comparisons. The correction is computed as follows:

$$\alpha' = \frac{\alpha}{m} = \frac{0.05}{21} \approx 0.00238$$

where α is the original significance level (typically 0.05), and m is the number of independent tests (in our case, 21 questions). Applying the Bonferroni correction to our results yields the adjusted outcomes shown in Table 4.3.

Table 4.3: Mann–Whitney U Test Results (Corrected)

Q#	U-Test	p-Value	Significant?
1	234.0	0.0291	Maybe
2	257.5	0.1053	No
3	267.0	0.2957	No
4	295.5	0.5230	No
5	248.5	0.1432	No
6	335.0	0.7151	No
7	382.0	0.3714	No
8	340.0	0.8550	No
9	311.0	0.3389	No
10	258.5	0.0780	No
11	283.5	0.3366	No
12	307.0	0.6245	No
13	235.5	0.0321	Maybe
14	146.0	0.0057	Maybe
15	237.5	0.0278	Maybe
16	266.5	0.1393	No
17	351.0	0.9678	No
18	256.0	0.1000	No
19	128.0	0.0000	Yes
20	118.0	0.0000	Yes
21	257.0	0.1850	No

In Table 4.3, the label "Maybe" represents borderline statistical significance under the adjusted rejection threshold (α'). Section 4.1.2 will discuss what this means, and how to improve the results.

We have evaluated our results to determine whether there were improvements in specific laboratory features. However, we may also wish to examine whether students generally expressed positive or negative attitudes toward these features. Instead of comparing the end-of-semester results to those of the "legacy" cohort, we can conduct an internal non-parametric analysis within the SPHERE cohort itself. This strengthens the interpretation of our findings by revealing which features students most favored, even in cases where many items show "no significant difference" (between cohorts).

It is important to note that not all questions are expected to yield a positive effect. For example, Question 21 ("I felt lost in the labs") would ideally produce a negative response, since lower ratings on this item reflect a more desirable outcome.

To assess this, a paired t-test could be considered. However, because the data are ordinal and not normally distributed, a non-parametric approach is more appropriate. Therefore, we use the one-sample Wilcoxon signed-rank test, which evaluates whether the median of the sample differs from a hypothesized value. Since student responses were collected on a 1–10 Likert scale, we set the hypothesized median to 5.5, representing a neutral midpoint. The hypotheses are defined as follows:

$\mathbf{H_0}$: The median student response does not differ from the neutral value ($\tilde{X} = 5.5$).
$\mathbf{H_1}$: The median student response differs from the neutral value ($\tilde{X} \neq 5.5$).

Since the direction of the students' responses (positive or negative) was not assumed in advance, a two-sided Wilcoxon signed-rank test was first performed to determine whether the median response for each question significantly deviated from the neutral midpoint. The direction of the effect was then interpreted from the positive and negative rank sums (W^+ and W^-) generated by the test. Significance was determined using the two-sided p-value; if the null hypothesis was rejected, the larger of the two rank sums indicated the direction of the shift. Specifically, if $W^+ > W^-$, responses tended to be above the neutral value (positive effect), whereas $W^+ < W^-$ indicated a negative effect.

Because 21 separate Wilcoxon tests were conducted—one for each survey question—we must account for the multiple comparison problem, which increases the risk of Type I errors. Since each question measures a distinct element from the labs, we applied the Bonferroni correction to maintain a family-wise error rate of 0.05. Thus, the adjusted significance threshold was set to $p < 0.00238$. The corrected results, including the direction of the effects, are summarized in Table 4.4. The table shows that students showed favorable results towards all questions, besides Q3 and Q21, which are to be expected. We do not want a positive effect in these questions, as it would indicate that students performed poorly while working on our labs.

Table 4.4: Wilcoxon Signed-Rank Test

Q#	Median	p-value	W^+	W^-	Effect
1	9.00	5.54e-11	1743.00	27.00	Positive
2	9.00	1.91e-10	1717.00	53.00	Positive
3	6.00	0.23	1043.50	726.50	No Effect
4	8.00	1.30e-07	1580.00	190.00	Positive
5	9.00	7.33e-10	1691.50	78.50	Positive
6	10.00	2.32e-10	1704.50	65.50	Positive
7	7.00	2.38e-06	1507.00	263.00	Positive
8	9.00	4.33e-10	1699.50	70.50	Positive
9	9.00	4.45e-10	1698.50	71.50	Positive
10	7.00	1.86e-04	1378.00	392.00	Positive
11	7.00	2.10e-06	1510.50	259.50	Positive
12	8.00	3.92e-08	1609.00	161.00	Positive
13	8.00	7.08e-09	1647.00	123.00	Positive
14	8.00	6.97e-10	1696.50	73.50	Positive
15	8.00	3.26e-08	1613.50	156.50	Positive
16	9.00	1.79e-09	1674.50	95.50	Positive
17	8.00	1.25e-08	1635.00	135.00	Positive
18	8.00	1.73e-07	1571.00	199.00	Positive
19	10.00	7.74e-12	1755.50	14.50	Positive
20	10.00	4.87e-12	1765.00	5.00	Positive
21	4.00	0.01	533.00	1237.00	No Effect

4.1.3 Lab Comprehension Results

Previously, we examined the results of the new lab features. As described earlier, those statistics were derived from an end-of-semester survey in which students rated their experiences on a Likert scale ranging from 1 to 10. In this section, we evaluate the pre- and post-lab results to determine whether students demonstrated measurable learning gains from the lab content. Once again, the Mann–Whitney U Test is used to assess statistical significance.

Unlike the survey data, the pre-/post-lab evaluations do not use a numeric scale. Students provided open-ended responses, which required us to establish a quantifiable

metric. Each response was assigned a value from 0 to 3, where 0 represents "completely incorrect" and 3 represents "fully correct." A smaller scale was selected to facilitate consistent evaluation of qualitative responses. After assigning these values, the Mann–Whitney U Test was applied to compare pre- and post-lab results.

Before stating the hypotheses, two important notes should be made regarding how these results are interpreted in Chapter 5:

- Because students' responses were anonymous, it was not possible to track individual performance. Thus, comprehension was evaluated at the class level rather than per student.
- These results assess comprehension gained from the newly developed labs only; they do not include legacy labs discussed in Section 4.1.2.

The hypotheses for this analysis are as follows:

H_0: There is no difference in students' comprehension before and after completing the lab.
H_1: There is a difference in students' comprehension before and after completing the lab.

The results of the Mann–Whitney U Test for both the Fall and Spring datasets are summarized in Table 4.5 and Table 4.6, respectively. Like before, we test at a confidence level of 95%.

Table 4.5: Mann–Whitney U Test Results (Fall 2024)

Lab	**U-Test**	**p-Value**	**Significant?**
Intro	630.0	8.91×10^{-7}	Yes
POSIX	289.0	1.68×10^{-7}	Yes
Buffer	280.0	4.90×10^{-6}	Yes
Pathname	532.5	3.30×10^{-2}	Yes
SQL Injection	296.5	2.41×10^{-6}	Yes
XSS	357.5	2.56×10^{-3}	Yes
Firewalls	257.5	1.03×10^{-5}	Yes

Table 4.6: Mann–Whitney U Test Results (Spring 2025)

Lab	**U-Test**	**p-Value**	**Significant?**
Intro	258.0	2.18×10^{-4}	Yes
POSIX	71.5	2.82×10^{-9}	Yes
Buffer	190.5	3.94×10^{-4}	Yes
Pathname	276.0	1.56×10^{-2}	Yes
SQL Injection	183.0	7.70×10^{-5}	Yes
XSS	187.0	5.08×10^{-5}	Yes
Firewalls	205.0	8.48×10^{-4}	Yes

Chapter 5

Discussion

This chapter will interpret the results found in Chapter 4, and reflect upon them.

5.1 Result Summary

5.1.1 Lab Feature Results

The Mann–Whitney U test showed that two out of our 21 questions were statistically significant indicating that two of our new lab features positively impacted students' overall experience. While this result falls below our expectations, there are important caveats associated with the Mann–Whitney U test, one of which was previously discussed: Our sample sizes make the test unstable. Although the test does not require the two populations to be of equal sample size, it performs better when the sample sizes are approximately the same [28].

Another issue with the test, mentioned in Section 4.1.2, is that conducting multiple comparisons between groups increases the likelihood of a Type I error—incorrectly rejecting a null hypothesis when it is actually true. To reduce this risk, we applied the Bonferroni correction, which aims to produce more reliable results from the data. A likely reason for our limited number of significant findings is the small sample size of students who completed the "legacy" labs. This issue will be discussed in Section 5.2.

One potential mitigation strategy we considered was to combine similar questions.

By reducing the number of comparisons, we could lessen our dependence on the Bonferroni correction. However, this approach led to a loss of specificity in the data. For example, Question 4 (labs provided necessary information to complete required work) and Question 12 (introductory text prepared students for the lab) both addressed the written materials provided in the labs. Combining them raised two concerns:

1. Students might feel differently about each question—they could favor one while disliking the other.

2. If the combined question were found to be statistically significant, it would be unclear whether Question 4 or Question 12 (or both) contributed to that significance.

Despite these limitations, we can confidently conclude from our results that students appreciated knowing their exact grades prior to submission and considered the auto-submission mechanism to be a useful new feature. With a larger dataset, some results that were marginally significant ("Maybe" significance) may attain stronger statistical support. These findings suggest that G1 and G2 were partially achieved, as outlined in Section 1.2.1.

Additionally, there are several valid reasons why some of these features may not be significant. Some students become accustomed to how these lab features work and may not remember parts of the lab that they find annoying. Moreover, some students have a general preference for how they complete their homework. For example, some students may prefer receiving all instructions at once, which is something they are typically used to by the time they take this course.

The highest p-value score is for Question 17, which asked whether the tips and warnings helped students understand what to do. For many students, the text may have been clear enough that they did not feel the need for additional tips or warnings, and therefore, did not use this feature. Another high p-value score is for Question 8, which asked whether students had enough time to complete the lab. This issue was not necessarily related to the labs themselves but rather to the flexible deadlines provided by the instructors. Therefore, it is difficult to quantify this consistently across semesters.

To provide stronger evidence of significance, additional data collection across future semesters would be beneficial. Increasing the sample size could enhance our confidence in

determining whether the revised lab design genuinely improved the students' experience.

Upon evaluating Table 4.4, we observe that all questions, except for Q3 and Q21, produced a positive effect. Since this test removes the comparison of our new labs with the "legacy", we are now able to assess students' perceptions directly within the new SPHERE lab environment. Each positive result indicates that students responded favorably to those aspects of the course, suggesting that the corresponding lab features were generally well received.

The two questions showing no significant effect (Q3 and Q21) correspond to items where a positive correlation was not necessarily desirable. Specifically, Question 3 stated, "I frequently needed to ask for help", and Question 21 stated, "I frequently felt lost while working on the labs". For these items, a negative rank would indicate improvement, as lower agreement reflects better student confidence and understanding. However, the lack of significance implies that the median student response did not differ from the neutral midpoint, meaning students neither agreed nor disagreed strongly with these statements. This outcome is reasonable within a learning context: if the labs were too difficult, we would expect a strong positive rank (indicating greater difficulty), whereas if they were too easy, we would expect a strong negative rank. The neutral result therefore suggests that the level of challenge was appropriate overall.

5.1.2 Lab Comprehension Results

Our results indicate that students successfully learned the material presented in the lab. Although these tests did not demonstrate a direct improvement associated with the lab features, they provide statistical evidence that students achieved meaningful learning outcomes. While the results are generally self-explanatory, it is worth noting that the Pathname lab—although statistically significant—had the highest p-value among the tests. Several factors may explain this finding.

First, students were introduced to directory traversal concepts in the Intro lab. Although the Intro lab did not explicitly teach exploitation techniques, it familiarized students with reading pathnames and navigating files and directories in a UNIX environment. Second, the relatively high p-value may be due to the simplicity of the Pathname lab material. As discussed in Section 3.1.1, this lab was designed to be quickly understood. Given that many students likely already possessed prior knowledge

of file paths, they were able to grasp these concepts more readily. These results show that G6 was achieved in Section 1.2.1.

5.1.3 Student Impact

Although this section does not include a specific metric value, students reported that these labs were presented in a distinctive and engaging way. While the labs were distributed digitally, some students chose to complete them using tablets with external keyboards. This demonstrates the flexibility and accessibility of the lab design, supporting Goal G2 described in Section 1.2.1.

As discussed in Section 2.3.1, students were previously required to use a combination of the `cp` and `scp` commands to submit their work. While the `cp` command was generally straightforward, many students found the `scp` command less familiar and more challenging to use. In the updated lab structure, students receive a clearer introduction to `scp` in the Intro lab but are not required to use it repeatedly throughout the semester. The Intro lab also functions as a practical reference guide for students who are new to Linux, allowing them to revisit the material as needed for support during the course.

5.2 Limitations

Throughout this study, several challenges were encountered that affected progress in production, evaluation, and lab delivery.

One of the main limitations was that these labs were developed concurrently with the SPHERE project. Since SPHERE is a four-year project currently in its beta phase, there were stability issues with the testbed including outages. Early in the study, we experienced performance issues with the labs, although these improved over time. Additionally, frequent updates to SPHERE often unpredictably disrupted features within our lab design. Some updates required only documentation changes, while others necessitated modifying scripts to maintain functionality. Examples of these updates include:

- The sidebar being redirected to different directories. Initially, it pointed to the students' home directory, then to a specific project/notebook directory, and later back to the home directory. This required updates to both the documentation and installation scripts.

- XDCs being rolled back, which removed some necessary extensions required for the notebooks.
- XDCs automatically expiring after four weeks unless an extension was manually added.
- Updates to JupyterLab that required students to sign into SPHERE a second time when accessing their XDC.
- Notebooks being executed under students' accounts instead of `root`, which required permission adjustments for certain scripts.
- Erroneous input randomly appearing in the terminal, caused by a bug in SPHERE that has not yet been identified (Figure 5.1).

Figure 5.1: Terminal glitch in SPHERE

Another limitation, common in large-scale applications, was the presence of bugs. Although the labs required hundreds of development hours, students often provided input that the notebooks were not designed to handle. A notable example of this occurred in the Buffer Overflow lab. Students received an introduction or review of C programming and were asked to add two numbers using variables. There were multiple valid ways to complete this task. For instance, students could add the two numbers within a variable and then print the variable using a `printf()` statement, or they could print the sum directly within the `printf()` statement without assigning it to a variable. During the first semester (Fall 2024), these inconsistencies caused instability, and the labs had to be rapidly updated to avoid hindering student learning. Fortunately, based

on the results in Table 4.5, this did not appear to significantly affect learning outcomes.

A limitation of our comparison tests is that the "legacy" labs were more polished than the new, "banditized" labs. While the "legacy" labs were in many ways less sophisticated, they were used for a decade, and had more development time. The new labs were being updated and refined while students were still taking them. As students discovered different ways to answer the same questions, these additional cases were incorporated into the notebook's scripts. With the labs in the first semester being unstable, the labs in the second semester were much more fine-tuned. This difference likely influenced some of the results.

A final limitation was the lack of communication with alumni. To obtain the "legacy" lab results, we were required to file an HRP-503 form before collecting human data. This process was completed midway through our timeline, after students had already completed the legacy labs and as we were preparing the new SPHERE lab delivery. Once HRP approval was granted, we attempted to contact former students for feedback. However, this proved difficult, as many had graduated and no longer checked their university email accounts. Furthermore, it had been nearly a year since they had completed the labs. We considered physically mailing survey forms, but it was likely that the students may have moved away after graduation. The university had recently implemented a policy that deactivates student email accounts shortly after graduation—a change introduced one month before our HRP submission. Unfortunately, due to these constraints, we had limited communication with our "legacy" cohort.

5.3 Personal Reflection

While this was a lengthy development process, it was ultimately very rewarding to deploy the labs and gather student feedback to further improve them. There were numerous challenges—not only those described in Section 5.2—but also a significant amount of innovation was required to make these labs engaging and different from standard computer science coursework.

If the development for this thesis were to restart, there are several changes I would consider. One major change would be to "abstract" the design as early as possible. Frequently, when I noticed that changes had to be made across multiple notebooks, it

was tedious to go through all seven notebooks and apply the same updates. I eventually learned how to use the `sed` command to quickly apply these patches via text replacement. Later into the development, these notebooks' design became abstracted to make debugging and patching easier. However, since this development didn't follow a standard software development methodology, it helped me understand why approaches like Agile are critical for large-scale projects.

Given the scale of this project, I plan to continue its development after completing my graduate program. Since this was an independent project with many design choices that would be difficult to teach another developer, continuing the work post-graduation is important. Once SPHERE's development is complete, the labs should be stable enough that they no longer require close supervision to prevent functionality from being broken by future updates. For those that wish to utilize this framework, documentation has been written to aid those who wish to leverage this design.

5.4 Future Work

For future work, we may consider extending these labs to accompany lecture material by providing hands-on tutorials for instructors to distribute during demonstrations. By utilizing SPHERE and Jupyter Notebooks, this approach could create a more interactive classroom environment and increase student engagement. Similar to the current labs, these tutorials could be preconfigured so that students would only need to run a script to prepare their Jupyter environment and initialize the sandbox.

In addition, developing more lab material would be a valuable direction for future work. Some labs previously hosted on DeterLab—but not used at UMD—have already been ported to their SPHERE equivalents. These labs use Jupyter Notebooks but do not yet follow the scaffolded learning approach. Examples include labs on password cracking, man-in-the-middle attacks, and worms.

Although the current labs are functional and effective, some components could benefit from further clarification and refinement. During the third semester of lab delivery, the main issues reported involved unclear instructions or confusion about certain features (such as saving and loading labs). This may be partially attributed to the transition to new teaching assistants. Since these labs are still relatively new, instructors and

teaching assistants require training on setup procedures, troubleshooting, and how to effectively communicate lab features to students. As the labs continue to be delivered over subsequent semesters, they will likely become more intuitive to use and easier to teach.

Another consideration would be to break apart some of these labs so they can be reused in other contexts. This approach would be similar to the original scaffolding proposal, creating a "dependency tree" of lessons, where new modules are unlocked only once previous requirements are completed. Since the current labs demonstrate how such a process could be implemented, this framework could be used for grant proposals for creating curriculum for cybersecurity education.

Finally, additional features could be incorporated into these labs in the future, depending on the progress of the SPHERE project. Once SPHERE completes its development phase, more advanced and stable functionality can be introduced without concerns about system instability.

Chapter 6

Conclusion

We have explored a new method of introducing cybersecurity concepts to students in this study. The transition from DeterLab to SPHERE provided numerous new tools and opportunities for innovation, which we successfully leveraged to create original, hands-on exercises for students. Based on the results we collected, several of these features appear to enhance students' comprehension, and students responded positively to them. The labs provide a unique and interactive environment that distinguishes them from typical computer science courses.

Inspired by OverTheWire: Bandit [15], we modernized outdated labs into a more robust system that can be adopted by other universities for their own students. These enhancements not only benefit students but also support instructors by simplifying grading and reducing workload through the use of dynamic, interactive notebooks. Even when students require instructor assistance, the step-by-step structure enables instructors to identify and resolve issues more efficiently. Compared to traditional approaches, this method of incremental learning and troubleshooting offers a more streamlined and effective teaching and learning experience.

With seven hands-on lab exercises, we have already expanded the material to include additional, previously outdated DeterLab content. Furthermore, we developed a generator that produces templates for instructors who wish to incorporate this system into their own course materials. The continued development of SPHERE has greatly improved accessibility for students, and the success of these labs is inherently tied to the success of the SPHERE platform.

Once SPHERE completes its development, it is expected that the scaffolded lab material will be further expanded and enhanced with new features. Because students have shown strong engagement and preference for this type of content, our goal is to ensure the labs remain as stable and effective as possible—maximizing their positive impact on students' learning outcomes.

References

[1] Yiyin Shen, Xinyi Ai, Adalbert Gerald Soosai Raj, Rogers Jeffrey Leo John, and Meenakshi Syamkumar. Implications of chatgpt for data science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2024, page 1230–1236, New York, NY, USA, 2024. Association for Computing Machinery.

[2] Edmund Pickering. Widespread usage of chegg for academic Misconduct. `https://doi.org/10.35542/OSF.IO/DS7YB`, n.d. Preprint. Accessed: 2025-03-13.

[3] B. Simon. Designing programming assignments to reduce the likelihood of cheating. In *Proceedings of the 19th Australasian Computing Education Conference (ACE '17)*, pages 42–47. ACM, 2017.

[4] Ramesh R. Naik, Maheshkumar B. Landge, and C. Namrata Mahender. A review on plagiarism detection tools. *International Journal of Computer Applications*, 125(11):16–22, September 2015.

[5] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Experience with deter: a testbed for security research. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006.*, pages 10 pp.–388, 2006.

[6] DETER Project. About deterlab, 2025. Accessed: 2025-03-18.

[7] Sphere Project. The sphere project: Humanitarian charter and minimum standards in humanitarian response, 2018.

[8] National Science Foundation (NSF). Nsf award #2330066: Research grant details, 2025.

[9] SPHERE Testbed Documentation. Sphere testbed overview, 2025.

[10] Jelena Mirkovic, Brian Kocoloski, and David Balenson. Enabling reproducibility through the SPHERE research infrastructure. *;login: Online*, 49(6), December 2024. Article shepherded by Rik Farrow.

[11] Project Jupyter. About project jupyter, 2025.

[12] Project Jupyter. Jupyterlab is ready for users, 2018.

[13] Lorena A. Barba, Lecia J. Barker, Douglas S. Blank, Jed Brown, Allen B. Downey, Timothy George, Lindsey J. Heagy, Kyle T. Mandli, Jason K. Moore, David Lippert, Kyle E. Niemeyer, Ryan R. Watkins, Richard H. West, Elizabeth Wickes, Carol Willing, and Michael Zingale. *Teaching and Learning with Jupyter*. Jupyter4Edu (GitHub), 2018. Open-access handbook for using Jupyter in teaching and learning.

[14] Jupyter Widgets Development Team. ipywidgets: Interactive widgets for jupyter, 2025.

[15] OverTheWire. Bandit wargame: A tutorial for beginners. `https://overthewire.org/wargames/bandit/`, 2025. Accessed: 2025-10-26.

[16] Jason H. Sharp. Using codecademy interactive lessons as an instructional supplement in a python programming course. *Information Systems Education Journal*, 17(3):20–28, June 2019.

[17] Christine Wania. College student perceptions of MyProgrammingLab and BlueJ in an introductory computing course. In *SAIS 2019 Proceedings*, 2019.

[18] Raymond Papp. Assessment of interactive e-learning platforms for business statistics and analytics. *Research in Higher Education Journal*, 46:1–11, 2025.

[19] Kateřina Dvořoková and Lumír Kulhánek. Innovation of the study course using pearson higher education tools. In *Proceedings of the CBU International Conference*

on Innovations in Science and Education, pages 593–598, Prague, Czech Republic, 2017.

[20] Wenliang Du. Seed: Hands-on lab exercises for computer security education. *IEEE Security & Privacy*, 9(5):70–73, 2011.

[21] Cynthia E. Irvine, Michael F. Thompson, Mark McCarrin, and James Khosalim. Labtainers: A docker-based framework for cybersecurity labs. In *Proceedings of the 2017 USENIX Workshop on Advances in Security Education*, 2017.

[22] Ambareen Siraj and Sazia Ghafoor. Crest-security knitting kit: Readily available teaching resources to integrate security topics into traditional cs courses. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 1058–1058, 2018.

[23] National Cybersecurity Training & Education Center (NCyTE). Ncyte educational materials. `https://www.ncyte.net/`, Accessed 2025.

[24] NICE Challenge Project. Nice challenges. `https://nice-challenge.com/`, Accessed 2025.

[25] OverTheWire Community. Overthewire wargames. `https://overthewire.org/wargames/`, Accessed 2025.

[26] Michael Thompson and Cynthia Irvine. Active learning with the cyberciege video game. In *4th Workshop on Cyber Security Experimentation and Test (CSET 11)*, 2011.

[27] Blue Team Labs Online, CloudLabs, and CYBRScore. Commercial cybersecurity lab platforms: Blue team labs, cloudlabs, and cybrscore. Blue Team Labs: `https://blueteamlabs.online/`; CloudLabs: `https://cloudlabs.us/`; CYBRScore: `https://cybrscore.io/`, Accessed 2025. Commercial cybersecurity training platforms.

[28] Paul–Christian Bürkner, Philipp Doebler, and Heinz Holling. Optimal design of the wilcoxon–mann–whitney-test. *Biometrical Journal*, 59(1):25–40, 2017.

[29] Dou Du, Taylor J. Baird, Kristjan Eimre, Sara Bonella, and Giovanni Pizzi. Jupyter widgets and extensions for education and research in computational physics and chemistry. *Computer Physics Communications*, 305:109353, 2024.