

Hacking the Learning Curve
Effective Cybersecurity Education at Scale

by

Connor David Nelson

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved February 2024 by the
Graduate Supervisory Committee:

Yan Shoshitaishvili, Co-Chair
Adam Doupé, Co-Chair
Ruoyu Wang
Tiffany Bao
Giovanni Vigna

ARIZONA STATE UNIVERSITY

May 2024

©2024 Connor David Nelson

All Rights Reserved

ABSTRACT

This dissertation introduces a comprehensive framework aimed at reshaping applied cybersecurity education to significantly ease the learning curve, at scale, through three synergistic innovations. These methods address the daunting educational barriers in cybersecurity, enabling learners at all levels to understand complex security concepts more easily.

The first innovation, the PWN methodology, redefines the traditional Capture The Flag (CTF) model by offering a structured series of modularized, self-guided challenges. This approach helps simplify complex topics into manageable units, each building on the last, which allows students to progress at their own pace. Over five years and with over 400 systems security challenges developed, this method has effectively helped students evolve from beginners to masters of advanced security exploits.

The second component is the DOJO platform, an open-source learning environment that uses containerization technology to provide a pre-configured, browser-based interface. This platform reduces the setup complexities associated with applied cybersecurity and has already given over 10,000 students immediate access to practical learning scenarios, from vulnerability discovery to advanced debugging, in a unified, user-friendly environment. Its seamless integration allows educators to quickly launch new challenges and resources, ensuring a continuous and dynamic educational experience.

The third component, the SENSAI tutor, is an AI-driven tutoring system that leverages Large Language Models to offer personalized, intelligent support. Integrated with the PWN methodology and DOJO platform, SENSAI serves as an on-demand mentor, providing tailored advice and problem-solving assistance. It adapts to individual stu-

dent needs, offering specific guidance and theoretical support to enhance understanding and retention of complex concepts.

Together, these three components create a powerful, integrated educational strategy that not only equips students with vital cybersecurity skills but also deepens their understanding of digital vulnerabilities and the strategic thinking needed to mitigate them. This strategy prepares a new generation of cybersecurity professionals to navigate the ever-evolving threats of the digital world.

ACKNOWLEDGMENTS

In my journey towards becoming a hacker (and what else would a PhD focusing on cybersecurity be!), I have been fortunate to have the support of so many people.

Thank you to my family, my day ones, who didn't stop me from pouring my entire childhood into my computer. It was my passion for video games, and in particular, creating "bots" to do the boring parts, that got me started on this path. Thanks to my mom, Kerri, a real finance hacker, who instilled in me a deeply meticulous, analytical mindset—who told me to get a job instead of doing a silly little PhD—who I ignored and did it anyway, and she still supported me nevertheless. Thanks to my dad, David, who taught me the importance of trying your best, and who set me down the path of cybersecurity, telling me cool stories about his buddies whose jobs were to hack into companies, and who brought me to DEF CON several times, introducing me to a world of hackers that I would soon be a part of. Thanks to my sister, Kaylee, my ride or die, who pushed me to one-up her Master's degree with a PhD—and I did, but I'm still not as cool as her. A special thanks to my grandma, Jean, and granny, Vicki, who have always supported me, and my grandpa, Mark, who fostered my love for programming as a child.

Thanks to Adam for starting me on my academic journey of hacking, when I sent him an email telling him about a vulnerability in his assignment submission server in November of 2016—and he replied, warning me that I could have been expelled for hacking the server, but instead invited me to take his cybersecurity class, join his cybersecurity club (the pwndevils), and eventually join his research lab, SEFCOM—thanks for not expelling me! It was this class that inspired me to create pwn.college, remembering the feeling of how cool it was to be hacking challenges for a grade, fighting my way to the top of the leaderboard (and getting there), and wanting

to share that experience with others. Thanks for all your “nerd snipes” (intellectual challenges that are all-consuming), and tricking me into a PhD by nominating me for—and ultimately being awarded—a Dean’s Fellowship award. Thanks for being a great mentor; teaching me that sometimes you just need to read the source code of PHP if you want to understand how it works; and showing me that hackers can be structured, organized, well-functioning members of society.

Yan. If Adam is the structure, Yan is the creative, inspiring chaos. Yan doesn’t know this, but the first time I saw him was at my first DEF CON in 2015, when my dad and I decided to go to his talk where he open-sourced angr, and then again the next year in 2016 when he led Shellphish to a top-3 finish in the DARPA Cyber Grand Challenge. The next time I saw him was at a pwndevils meeting, where we were play-testing the iCTF infrastructure. He walked into the room late, sat down, pulled out his laptop, and pwned the infrastructure in less than 5 minutes. On the job market, I remember hoping that Yan would decide to come to ASU, and being stunned when he actually did. I was (and still am) a “fan of Zardus”.

Yan remembers “tricking” me into being the teaching assistant (TA) for his first real class, CSE 466. I remember attending his talk at DEF CON in 2017, after he accepted his position at ASU, chatting with him in the hallway after to tell him that I was going to be his student, walking into his office after he started, and telling him that I was going to be his TA. It wasn’t that ASU assigned me to be his TA, it was that I assigned myself to be his TA. And it was one of the best decisions I’ve ever made. Yan is one of most interesting, inspiring, intelligent, and thoughtful people I’ve ever met. He has taught me so much, and encouraged me even more—without him, there would be no pwn.college, and I would not have finished my PhD. When I told him I wanted to quit my PhD, he immediately rejected the idea, and told me

that I was going to pivot my research to education. Yan is constantly coming up with new—mostly insane—ideas, and every once in a while, one of them is really, really good. Yan, thank you for the incredible journey you’ve taken me on.

Thank you to the other members of my committee—Fish, Tiffany, and Giovanni. Fish, your tireless work ethic is inspiring; and Tiffany, your ability to see the big picture is invaluable. Thank you Giovanni for being the person that started it all—without you, there would have been no Shellphish, no SEFCOM, and no pwn.college.

Thank you to Wil and Zion for all your thankless work in making CTF actually possible at ASU, leading both pwndevils, and then Shellphish. Thank you to Erik for your mentorship and countless unique and thoughtful perspectives. Thank you to Kyle for constantly scaring me with your hacker skills, fearing that pwn.college is going to get hacked any day now. Thank you to Robert for your dedication to pwn.college, wanting to make it be the best it can be, and constantly challenging me to do the same. Thank you to the rest of SEFCOM for the incredible research journey, and the rest of Shellphish for the incredible hacking journey.

Thank you to the pwn.college community. Thank you for your constant support, your feedback, and your enthusiasm—if it wasn’t for you, I would have never finished my PhD. There is truly nothing more rewarding than seeing someone learn something new. This is just the beginning!

To Julia Hack¹, the love of my life, my best friend, my (soon-to-be) wife. Thank you for your support, your love, and your patience. While you missed the start of this journey, you were absolutely crucial in its completion. Julia, if TAing for Yan was *one of* the best decisions I’ve ever made, you are *the* best decision I’ve ever made. I dedicate this dissertation to you. I love you.

¹Yes, Julia’s last name is actually “Hack”. If that isn’t fate, I don’t know what is!

TABLE OF CONTENTS

| CHAPTER | Page |
|--|------|
| 1 INTRODUCTION | 1 |
| 1.1 Capture The Flag | 2 |
| 1.2 The Learning Curve | 3 |
| 2 PWN THE LEARNING CURVE: EDUCATION-FIRST CTF CHALLENGES | 6 |
| 2.1 Introduction | 7 |
| 2.2 Related Work | 9 |
| 2.3 Individualized Concepts..... | 13 |
| 2.3.1 Shellcode Injection | 13 |
| 2.3.2 Reverse Engineering | 14 |
| 2.3.3 Memory Errors | 15 |
| 2.3.4 Return Oriented Programming | 16 |
| 2.3.5 Dynamic Allocator Misuse | 17 |
| 2.3.6 Sandboxing | 17 |
| 2.3.7 Race Conditions..... | 18 |
| 2.3.8 Kernel Security..... | 19 |
| 2.4 Integration Challenge Design | 19 |
| 2.4.1 Program Exploitation..... | 20 |
| 2.4.2 System Exploitation | 20 |
| 2.5 Discussion | 21 |
| 2.5.1 Smoothing Learning Curves | 21 |
| 2.5.2 Challenge Generation | 22 |

| CHAPTER | Page |
|---|------|
| 2.5.3 Refining Core CS Concepts..... | 23 |
| 2.6 Evaluation | 24 |
| 2.6.1 Survey | 24 |
| 2.6.2 Follow-on Course | 26 |
| 2.7 Conclusion | 28 |
| 3 DOJ0: APPLIED CYBERSECURITY EDUCATION IN THE BROWSER | 29 |
| 3.1 Introduction | 30 |
| 3.2 Related Work | 32 |
| 3.2.1 Capture The Flag Projects | 32 |
| 3.2.2 Wargame Platforms | 34 |
| 3.2.3 Cyber Range Projects | 34 |
| 3.2.4 Commercial Training Platforms..... | 36 |
| 3.3 Dojo Design | 37 |
| 3.3.1 Challenge Environments | 39 |
| 3.3.2 Working Environments..... | 40 |
| 3.3.3 Environment Access | 42 |
| 3.3.4 Challenge Virtualization | 43 |
| 3.3.5 Instructor Capabilities | 45 |
| 3.4 Discussion | 46 |
| 3.4.1 Deploying DOJ0 | 47 |
| 3.4.2 Supported Challenge Types | 47 |
| 3.4.3 An Open Deployment | 48 |
| 3.4.4 Rapid Learning Launch | 48 |

| CHAPTER | Page |
|---|------|
| 3.4.5 Scaling and Compute Requirements | 49 |
| 3.4.6 Impact of Feature Improvements | 49 |
| 3.5 Evaluation | 50 |
| 3.6 Conclusion | 52 |
| 4 SENSAT: LARGE LANGUAGE MODELS AS APPLIED CYBERSECURITY TUTORS | 53 |
| 4.1 Introduction | 54 |
| 4.2 Background and Related Work | 57 |
| 4.2.1 Large Language Models | 57 |
| 4.2.2 Cybersecurity Education | 59 |
| 4.2.3 LLMs in Education | 60 |
| 4.3 SENSAT Design | 62 |
| 4.3.1 SENSAT's Environment | 62 |
| 4.3.2 SENSAT's Context | 64 |
| 4.3.3 SENSAT's Prompting | 67 |
| 4.4 Widescale Deployment | 70 |
| 4.4.1 Full Session Example | 70 |
| 4.5 Quantifying SENSAT's Impact | 71 |
| 4.5.1 Impact of Terminal and File Context | 73 |
| 4.5.2 Performance Across Modules | 75 |
| 4.5.3 Impact of an Educational System Prompt | 78 |
| 4.6 Student Feedback | 81 |
| 4.7 Analysis and Discussion | 84 |
| 4.7.1 Importance of Context | 85 |

| | |
|---|------|
| CHAPTER | Page |
| 4.7.2 Unknown Unknowns | 87 |
| 4.7.3 Size of the Context Window | 87 |
| 4.7.4 Reflexive Intelligence | 88 |
| 4.7.5 Hallucinations | 90 |
| 4.8 Conclusion | 91 |
| 5 THE PWN.COLLEGE | 92 |
| 6 THE FUTURE OF CYBERSECURITY EDUCATION | 97 |
| REFERENCES | 99 |

Chapter 1

INTRODUCTION

The summer after I graduated high school, I attended DEF CON for the first time. My dad and I made the drive out to Las Vegas, and I was immediately taken aback by the unique atmosphere of hacker culture. I had never seen so many people with dyed hair, piercings, and tattoos in one place, all of whom were wearing ridiculous 7" vinyl records on lanyards around their necks, contrasting sharply with the gaudy casinos' artificial, freezing cold interiors, set against the sweltering heat of the Nevada desert in August. DEF CON is a truly bizarre environment that does not make sense.

We attended talks on topics ranging from remote exploitation of unaltered passenger vehicles [56], to modding Mattel children's toys to do radio frequency hacking in order to force-open garage doors [41], to the release of a binary analysis framework called *angr* [83]. These talks, while cool, weren't really all that interesting; I didn't understand most of what was being said, I only knew that it was cool. What was interesting though, was *Packet Hacking Village* [21], where I learned how to use Wireshark to analyze network traffic, and *warlock gam3z CTF* [22], where I learned how to use basic static analysis techniques to extract a flag from a binary. It was here that I was first introduced to the concept of Capture The Flag (CTF); it was very *real*, and I was hooked.

Eventually we stumbled upon the *DEF CON CTF* [20] room, where we watched a bunch of mostly-twenty-something-year-olds sitting in clusters around different tables, staring into the void of their laptops, sometimes saying something to someone seated near them, and—on rare occasion—yelling with excitement. There was a

scoreboard projected onto the wall with teams including **Shellphish** [82], **PPP** [76], and **DEFKOR** [27] all competing to be the best hackers in the world. One of the organizers was walking around, and explained to us that the teams were analyzing multiple binary services, patching them, and exploiting the other teams to gain points every few minutes, and that the visualization on the wall showed every time a team successfully hacked another team. Much like the talks, I didn't really understand what was going on, how their activity looked in comparison to what I had done in *warlock gam3z CTF*. I didn't know at the time, for example, that for every cluster of 8 people sitting together at the same table, there was another 40 people on their team working remotely from a nearby hotel suite, sitting in front of their own laptops, staring into their own voids. I didn't know why they were doing what they were doing, or how they were doing it, but I knew that it was cool. These were the best hackers in the world.

1.1 Capture The Flag

As I now know, the objective of Capture The Flag (CTF) is simple: capture the flag [25]. It's not a physical flag, but a secret blob of data, a string of text, called a "flag", that is a token of success. Often it looks something like `flag{b37_y0u_d1dn7_3xp3c7_a_f14g_1n_my_d1553r74t10n}`.

Once you have the flag—that exact piece of text—you submit it to the CTF scoreboard, and you get points. The challenge is that the flag is guarded by a program (often called the "challenge", or sometimes the "service"). The specifics of how the program guards the flag are different for every challenge, but the general idea is that the program isn't designed to just give you the flag, but it is possible because of some

vulnerability—or some non-obvious detail that must be figured out—in the program that you are able to get the flag. By getting the flag, you have proof that you have successfully exploited the vulnerability, or demonstrated that you understand the non-obvious detail, and you get points.

The smartest people I have ever met are CTF players. This format is *incredibly* conducive to learning, and it is the reason that I am a cybersecurity expert today. Staring at a problem for hours, with a team of people who are all doing the same, testing ideas, figuring out why those ideas didn't work, and then trying something else, forces you to *truly* understand. The thrill of getting a flag after hours of work is *incredible*.

At its core, CTF really has nothing to do with cybersecurity. It has to do with problem solving, figuring out how things work, how to *make* things work. It has to do with learning, with pursuing knowledge. At its core, it has to do with *hacking*. Not the type of “hacking” that is illegal, or is unethical, or has to do with breaking things, but the type of hacking that is about understanding, about making things work, about creating. The best hackers are the ones who are the most creative, the most innovative, the most curious—the ones that have the most perseverance and the most passion. The ones that have mastered *learning*. CTF is nothing more than an outlet for hacking—and hacking *is* learning.

1.2 The Learning Curve

Unfortunately for most people looking to get started, CTF does not have a learning curve—it has a learning wall. The traditional path from “I don't know anything about

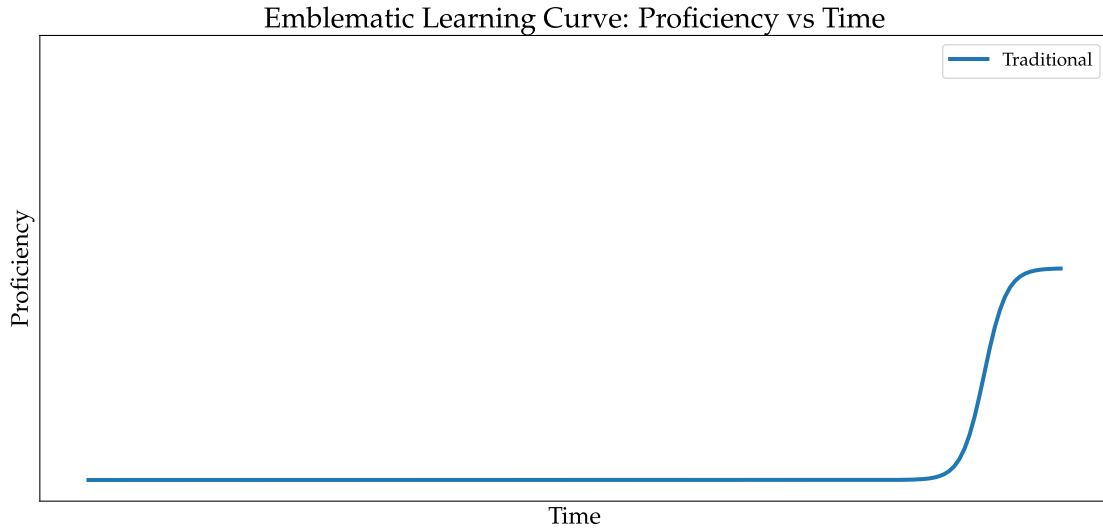


Figure 1. Emblematic Learning Curve: Traditional

cybersecurity” to “I’m a proficient hacker” looks something like Figure 1. Many novices will stop before they get anywhere near the wall.

I have first hand experience with this. Even as a member of one of the top (and *the* longest standing) CTF teams in the world, Shellphish, we have had a hard time getting new members to stick around. Getting them to the point where they can contribute to the team—and feel *useful*—is a long, daunting process through the learning abyss. Most do not make it.

But can something be done about this? What is incredible about CTF, is that for those who do make it, they are able to learn topics *deeply*. And even better, this is possible *at scale*. The cost of an additional student is nearly zero; a challenge is created, and then it can be worked on by any number of students.

In this work, I focus on overcoming the pedagogical limitations of CTF; so that more people can learn, so that more people can *hack*. First I introduce a novel methodology—PWN—for creating *educational* CTF challenges, which are designed to provide guidance and incrementally introduce students to new concepts. Next I build

a novel infrastructure—DOJO—for hosting and working on these challenges, accessible entirely from within the browser, so that students can begin *immediately* hacking without any barriers. Finally, I present a novel tutoring system—SENSAI—which is able to perceive what students are doing, and use that information in order to provide *automated*, immediate, individualized feedback and guidance to students, at scale. All of this comes together to create `pwn.college`²—to smoothen the learning curve—so that we can once and for all “*tear down this wall!*” [77].

²`pwn.college` is a *free* online cybersecurity education platform available at <https://pwn.college/> for anyone interested in learning how to hack!

PWN THE LEARNING CURVE: EDUCATION-FIRST CTF CHALLENGES

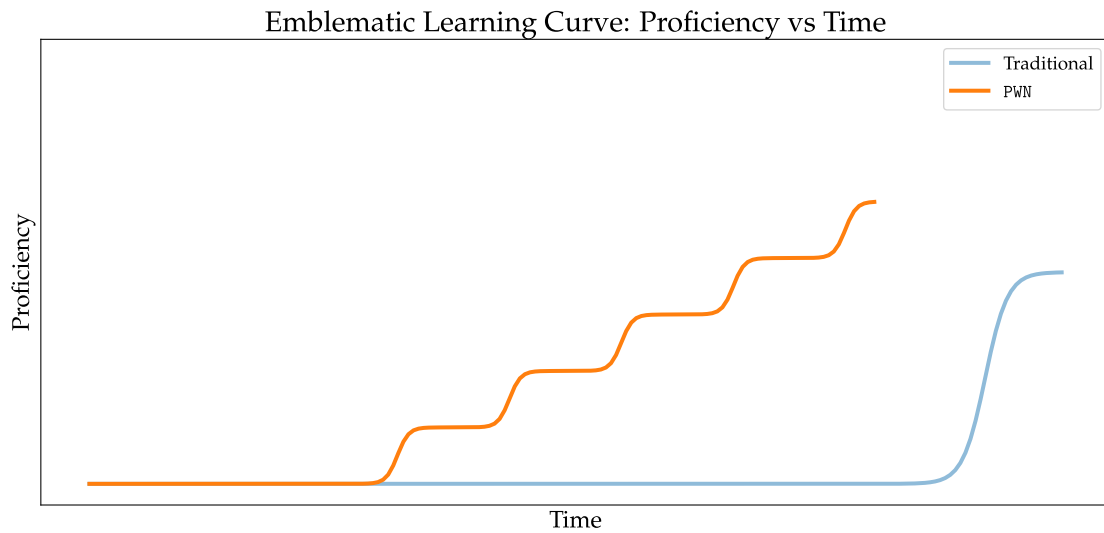


Figure 2. Emblematic Learning Curve: PWN

The PWN methodology breaks down the learning curve into a series of small, achievable steps. This is in contrast to the traditional CTF approach, which often presents a steep learning curve. We are able to achieve this by introducing a series of challenges which decompose the topic into smaller, more manageable challenges, each slowly building on the previous, and each providing a learning opportunity. This is further enabled through self-guiding, self-introspective challenges, which offer a detailed analysis of the interworkings of the challenge. In doing so, we are able to provide a more individualized learning experience, which is more effective and more engaging.

2.1 Introduction

Modern cybersecurity requires an understanding and integration of several concepts concurrently. For example, a software exploit might involve a race condition which enables a heap use-after-free. This primitive might cause memory corruption that, in turn, can be used to achieve arbitrary code execution through return oriented programming. If the target is a sandboxed process, OS kernel vulnerabilities (for example, a kernel race condition) might then be abused to gain code execution within the context of the kernel. Is it possible to teach all of these concepts to the average undergraduate computer science student? In order to understand this exploit, students must master reverse engineering, race conditions, dynamic allocator misuse, memory errors, return oriented programming, sandboxing, and kernel security.

Despite this complexity, conveying such understanding is critical. Apple will pay up to \$2 million for a zero-click remote chain that provides full kernel execution with persistence, includes a kernel PAC bypass, and circumvents the protections of Lockdown Mode [2]. Such exploits fetch this price because they are constantly being sought out, found, and used by bad actors to circumvent the security of people and organizations [98]. Effective education methodologies to instill in students an ability to identify and fix such flaws is critical to the security of the modern software ecosystem.

In this chapter, **we implement and discuss a novel methodology for teaching advanced cybersecurity concepts to mastery at scale**. We explore the strategies necessary for students to not only conceptually understand vulnerabilities, but to apply this knowledge by crafting advanced end-to-end exploits from scratch. Furthermore, we address the challenge of achieving this at scale, catering to hundreds or even thousands of students. By the end of a single semester undergraduate course at a

large public R1 research university, students—some of whom have previously never seen x86-64 assembly—craft end-to-end exploits which, for example, leverage kernel vulnerabilities to achieve arbitrary code execution in a sandboxed process with modern mitigations in place. Two crucial observations guide the design of our methodology.

First, we observe that **although security concepts must be integrated concurrently, they need not be taught concurrently or even atomically**. We explore individual cybersecurity concepts each in their own module, building them up gradually through a series of challenges. We begin with a basic and simplified problem and gradually introduce more advanced concepts and realistic constraints, progressively weaving through constituent concepts and gently introducing ideas that will be crucial to succeeding in later challenges. By minimizing conceptual jumps between challenges, we aim to flatten the learning curve and make the material more approachable. After the students master several concepts independently, we combine them.

Second, we observe that students **need not learn to apply a concept, while simultaneously learning the often-times complex introspection tools necessary to debug and understand why their solution is failing**. Students often struggle to appropriately employ tools like `gdb` and `strace` for aiding their understanding of the problem. This presents a massive learning barrier, as they have no way to know if their solution is failing because of a bug in their solution or a bug in their understanding of the concept. To solve this problem, we have developed *self-guiding* variants of all challenges. This allows students to understand the concept before moving on to the regular variant, which requires them to use their own analysis tools such as `gdb` to solve the challenge. This approach helps overcome difficulties of

performing introspection and debugging by decoupling them from the learning of the concept itself, and reduces barriers to progress for many students.

Our challenges draw inspiration from Capture The Flag (CTF), but we have observed that while CTFs are valuable, they are not a universal solution in cybersecurity learning: CTF challenges are designed to be challenging, not necessarily educational. Adopting an education-first approach, we created over 400 challenges over five years, tailored for our undergraduate Computer Systems Security course. These challenges have been integrated into a globally accessible educational website, teaching binary exploitation concepts to over 10,000 students. In this chapter, we detail our experience designing these challenges and discuss the lessons we have learned in the process: highlighting the design of the challenges, what worked, what didn't, and why. To evaluate our methodology's effectiveness, a survey was administered to students who participated in our course. The results showed that a majority found the challenges instrumental in grasping cybersecurity concepts. Following this, a subsequent course on vulnerability research indicated that students could leverage this foundational knowledge to identify and report vulnerabilities in real-world software. As a commitment to global education, we make all challenges and accompanying lecture materials discussed herein freely, and easily accessible, to the world, available at <https://pwn.college/>.

2.2 Related Work

Researchers have explored the use of hacking as a pedagogical tool to teach computer science and cybersecurity concepts. In [11], Bratus defines hacking by “the ability to question the trust assumptions in the design and implementation of computer

systems rather than any negative use of such skills.” Through the hacker lens, students are able to perceive a cross-layer view of computer systems. This approach encourages students to think critically and creatively about computer systems. By analyzing the failure modes of systems from a hacker’s perspective, students can gain a deeper understanding of how the various layers of a system interact and can be exploited.

Applied cybersecurity labs are an effective pedagogical approach for cybersecurity concepts such as SDN Security [71], Reverse Engineering [3], SQL Injection [6], and Android mobile security [44]. Moreover, researchers have proposed using hacking as toy problems to teach various underlying computer science concepts [64]. For example, parsing can be taught through the example of intrusion detection systems, file systems can be taught through forensics and recovering deleted files, and assembly and memory allocation can be taught through shellcode and buffer overflows.

Capture The Flag (CTF) events are popular in the cybersecurity community as opportunities for participants to apply their knowledge to solve complex challenges in a competitive format. The educational merit of hacking events has been acknowledged by the academic community, as evidenced by academic competitions like iCTF [90], picoCTF [15], and CSAW CTF [32] that specifically cater to students and enable them to showcase their abilities and engage in peer competition. CTF events have become an effective way to cultivate the next generation of cybersecurity experts [5, 28] and advance the state of the art in cybersecurity. Increasingly, CTF is being used within academic courses, with many educators incorporating them into their courses as exercises [57] or even making them the focal point of a flipped classroom model [14, 63]. Educators report increased student engagement, and improved self-confidence and motivation among the students [47, 26, 14].

CTF can be an incredibly valuable educational tool, but there are many challenges

and pitfalls, especially for novices. While a flag-based system provides instant feedback on the correctness of an answer, it lacks partial flags or credit, making it difficult to gauge progress. This limitation is especially pronounced in challenges intentionally designed to frustrate competitors, which are unsuitable for educational purposes. Instead, a well-designed challenge should guide students through the solution process, as Chung argues [17]. However, a methodology for designing challenges that strike the right balance between being engaging and informative is challenging and has received limited attention in the literature. In this chapter, we focus on addressing these limitations.

Vykopal et al. [91] identified three main reasons why beginners may become discouraged: overly difficult challenges, ambiguous challenges, and limited feedback on progress. In [96], the authors aimed to target a novice population by designing challenges that did not require significant prior knowledge (for example, avoiding compiled binaries, which were considered too difficult). However, novices still struggle to self-learn and require guidance and support. For instance, in [58], researchers provided novices with mentors and taught a lecture on the networking tool `tcpdump` to prepare them for upcoming challenges. The authors of picoCTF found that competitors tend to prefer challenges that require minimal familiarity with the command line or additional tools, and consequently stressed the importance of teaching tooling to students [15]. Similarly, [95] found that while there is no way to fake the knowledge needed in a CTF challenge, the game must be at an appropriate level for the audience. If students have not studied networking and lack experience with the command line, they may experience difficulty and frustration. As a solution, the authors suggest creating tutorials and “level zero” versions of challenges to provide a foundation for students to build upon. This approach allows novices to gradually develop their

skills and knowledge, gaining confidence and motivation as they progress through the challenges. In this chapter, we expand upon this idea of a “level zero” version of challenges, by providing an entire progression of challenges that build upon each other to ramp up the difficulty and complexity, and continue to build up confidence and knowledge along the way.

In order to gain better insight into how students approach and progress in solving challenges, some tools have been developed which introspects their process [94, 59]. These tools help educators identify challenging tasks and improve educational materials. In this chapter, we focus specifically on the process by which we enhance those educational materials, and in particular, enhance the challenges. We take the insight of where a student is struggling and use it to guide the design of predecessor challenges that will help them succeed.

Designing effective challenges can be both a difficult and time-consuming task. Alpaca [29] is a system that uses AI and a database of vulnerabilities to generate challenge scenarios automatically according to user-specified constraints. The generated scenarios include a series of vulnerabilities and exploits that must be overcome to complete the challenge and create virtual machines with these vulnerabilities built-in. Users can specify complexity levels and specific vulnerabilities that must be used. Conversely, SecGen [80] generates challenges by randomly selecting composable modules. In [31], the authors automatically generate variants of reverse engineering challenges in order to prevent cheating. picoCTF also focuses on automatic challenge generation [13], but uses a much more trivial find-and-replace approach against challenge values. In this chapter, we focus on templated challenge generation which enables very slight modifications within low level program details. Rather than using different high level vulnerabilities present within a scenario, or focusing only on the task of replacing

specific values in a challenge, as done in prior work, our templating methodology is more generic and allows us to easily create a progression of challenges that build upon each other by specifying only the differences between challenges. This provides a significant advantage over the prior work, as it allows our challenges to be much more maintainable, and is what enables challenge progression in the first place.

2.3 Individualized Concepts

We split the complex area of binary security into a number of single-concept modules, which we detail in this section. Because Linux-based infrastructure is easier to build and maintain, our challenges explore these concepts in a Linux environment, although analogues can conceptually be created for other platforms as well. Successful solution of a challenge allows students to retrieve challenge-specific secret files, called *flags*, that they can redeem for credit in our institution’s Computer Systems Security course.

2.3.1 Shellcode Injection

Shellcoding is the art of injecting code into a program, usually during exploitation, to induce attacker-desired actions. It is traditionally taught by presenting students with a stack buffer overflow and an executable stack, and expecting students to figure out memory corruption, an exploit payload, as well as learning how to debug their exploit all at once. Instead, this module explicitly focuses on the payload by ingesting shellcode and immediately executing it, allowing students to fully focus on learning about shellcode itself. Throughout the module, we gradually introduce additional

constraints on this shellcode to force students to reason creatively about accomplishing attacker goals under difficult situations.

Before running shellcode, teaching variants disassemble the ingested payload, after any mutations and constraints are applied, and display it. Additionally, we have recently implemented an automatic breakpoint at the beginning of the shellcode when students start the challenge from a debugger, to ease troubleshooting. While students can, of course, do both of these things manually, simplifying the debugging process helps avoid frustration.

2.3.2 Reverse Engineering

Reverse Engineering is the process of analyzing a system in order to derive knowledge of its design and implementation. While this often means simply learning to read assembly, in order to truly master reverse engineering, it is important to be able to infer and understand the core design of a target system. This understanding, in turn, is typically used to reason about vulnerabilities in binary code. In this module, however, we free students from the need to reason about vulnerabilities by creating challenges requiring students to understand and invert algorithms that are implemented with (through the course of the module) increasing amount of obfuscation.

Teaching variants display the results of data mutations as they occur, allowing students to double-check their hypotheses. In later levels, teaching variants display the VM state and executed instructions to enhance initial student understanding.

```

Send your payload (up to 8 bytes)!
AAAAAAA
You sent 8 bytes!
Let's see what happened with the stack:
+-----+-----+-----+
| Stack location | Data (bytes) | Data (LE int) |
+-----+-----+-----+
| 0x00007ffc82475420 (rsp+0x0000) | 00 00 00 00 00 00 00 00 | 0x0000000000000000 |
| 0x00007ffc82475428 (rsp+0x0008) | 08 66 47 82 fc 7f 00 00 | 0x00007ffc82476608 |
| 0x00007ffc82475430 (rsp+0x0010) | f8 66 47 82 fc 7f 00 00 | 0x00007ffc824765f8 |
| 0x00007ffc82475438 (rsp+0x0018) | a0 86 86 93 01 00 00 00 | 0x000000001938686a0 |
| 0x00007ffc82475440 (rsp+0x0020) | 24 87 86 93 29 7f 00 00 | 0x00007f2993868724 |
| 0x00007ffc82475448 (rsp+0x0028) | 08 00 00 00 00 00 00 00 | 0x0000000000000008 |
| 0x00007ffc82475450 (rsp+0x0030) | 41 41 41 41 41 41 41 41 | 0x4141414141414141 |
| 0x00007ffc82475458 (rsp+0x0038) | 00 00 00 00 00 00 00 00 | 0x0000000000000000 |
| ... | | |
| 0x00007ffc824754b8 (rsp+0x0098) | 00 00 00 00 00 00 00 00 | 0x0000000000000000 |
| 0x00007ffc824754c0 (rsp+0x00a0) | d0 11 40 00 08 00 00 00 | 0x00000008004011d0 |
| 0x00007ffc824754c8 (rsp+0x00a8) | 50 54 47 82 fc 7f 00 00 | 0x00007ffc82475450 |
| 0x00007ffc824754d0 (rsp+0x00b0) | 00 65 47 82 fc 7f 00 00 | 0x00007ffc82476500 |
| 0x00007ffc824754d8 (rsp+0x00b8) | c7 1f 40 00 00 00 00 00 | 0x0000000000401fc7 |
+-----+-----+-----+
The program's memory status:
- the input buffer starts at 0x7ffc82475450
- the saved frame pointer (of main) is at 0x7ffc824754d0
- the saved return address (previously to main) is at 0x7ffc824754d8
- the saved return address is now pointing to 0x401fc7.
- the address of win() is 0x401833.
If you have managed to overwrite the return address with the correct value,
challenge() will jump straight to win() when it returns.
Let's try it now!

```

Figure 3. Memory Errors teaching variant.

2.3.3 Memory Errors

Memory errors occur when a user is able to corrupt memory they're not supposed to, the result of which can have brutal effects and may enable significant control over a program. To understand this concept requires an understanding of a program's runtime as it uses control flow metadata, common issues surrounding buffer overflows and buffer overreads, as well as the impacts of static and dynamic memory layouts. Over the course of this module, we guide students from a simple overflow into a boolean variable through a number of complex concepts that require students to reason about security mitigations, exploitation in partial-knowledge settings, and other concepts. To reduce the number of concepts they have to consider, we provide a "win" function for students to eventually redirect execution to, rather than forcing them to reason about shellcode.

Teaching challenges in this module display critical memory data, especially control data stored on the stack such as return addresses, canaries, and saved registers, as shown in Figure 3. While this information is accessible through a debugger, having

```

$ echo -ne "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" | /challenge
Received 160 bytes! This is potentially 5 gadgets.
Let's take a look at your chain!
Note that we have no way to verify that the gadgets are executable from within this challenge.
You will have to do that by yourself.
+--- Printing 6 gadgets of ROP chain at 0x7ffed754998.
| 0x00000000000402709: pop rax ; ret ;
| 0x000000000000003c: (UNMAPPED MEMORY)
| 0x00000000000402720: pop rdi ; ret ;
| 0x000000000000002a: (UNMAPPED MEMORY)
| 0x00000000000402738: syscall ; ret ;
| 0x0000000000000000: (UNMAPPED MEMORY)
Leaving!
$ echo $?
42

```

Figure 4. Return Oriented Programming teaching variant.

it readily available allows students to focus on understanding the concept first, then understanding the debugger later.

2.3.4 Return Oriented Programming

Return Oriented Programming (ROP) is a *code reuse* attack in which attackers redirect control flow not to injected shellcode (for example, because Data Execution Prevention mitigations leave no place for executable shellcode), but to a series of disjoint sets of instructions (termed *gadgets*), with control flow transfers facilitated by `ret` instructions coupled with attacker control of the program stack. Throughout this module, we guide students from first calling a single function, to chaining multiple functions, and gradually through the chaining of ROP gadgets in complex scenarios.

Analogous to Shellcode Injection, teaching challenges disassemble the ROP chain that would be executed before executing it. This is a feature not seen even in popular debuggers, and helps students understand what is happening when their ROP chain executes.

```

This challenge can manage up to 1 unique allocations.
[*] Function (malloc/free/puts/read_flag/quit): malloc
Size: 407
[*] allocations[0] = malloc(407)
[*] allocations[0] = 0x5630d7b702c0
[*] Function (malloc/free/puts/read_flag/quit): free
[*] free(allocations[0])
=====
| TCACHE BIN #24 | SIZE: 393 ~ 408 | COUNT: 1 | HEAD: 0x5630d7b702c0 | KEY: 0x5630d7b70010 |
|-----|
| ADDRESS | PREV_SIZE (-0x10) | SIZE (-0x08) | next (+0x00) | key (+0x08) |
|-----|
| 0x5630d7b702c0 | 0 | 0x1a1 (P) | (nil) | 0x5630d7b70010 |
|-----|
[*] Function (malloc/free/puts/read_flag/quit): read_flag
[*] flag_buffer = malloc(407)
[*] flag_buffer = 0x5630d7b702c0
[*] Read the Flag!
[*] Function (malloc/free/puts/read_flag/quit): puts
[*] puts(allocations[0])
Data: FLAG{...}

```

Figure 5. Dynamic Allocator Misuse teaching variant.

2.3.5 Dynamic Allocator Misuse

In recent years, attackers have bypassed exploitation mitigations by abusing dynamic memory allocators to achieve full control of program memory. This has historically been considered a somewhat secret art, and mostly explored in high-skill CTFs. To keep things tractable for students, this module mostly focuses on the details of the implementation of the `tcache` allocation caching layer of Linux’s `glibc`, attacks against which are simpler than the general case of allocator misuse.

The teaching variants display the current state of `tcache`, including cached allocations and any relevant metadata, as shown in Figure 5. While, again, there are tools that can perform this function, including this in the challenge enables students to tackle the problem head on without any impediments.

2.3.6 Sandboxing

Sandboxing is the practice of isolating code to limit attacker capabilities during exploitation. While many courses discuss this concept within systems security, we have found few that actually introduce labs or challenges which practically explore

it. To maximize ease of learning, this module assumes arbitrary code execution has already been achieved (by directly running the shellcode that students have learned to write in the previous module) in order to focus entirely on the sandboxing mechanism itself. The specific sandboxing mechanisms used evolve over the course of the module, exposing students to different sandboxing scenarios.

The teaching variants describe the jailing process as it happens, describing the resources that are available at what point of challenge execution.

2.3.7 Race Conditions

Concurrency errors are some of the trickiest and least-understood software bugs. Because they occur in many different contexts and are dependent on the specifics of the software in question, the teaching of concurrency errors seems to be less common than the teaching of other software errors. Our challenges progress from filesystem-based time-of-check to time-of-use (TOCTOU) attacks into complex mutex issues in memory, allowing students to gradually familiarize themselves with underlying concepts.

As challenges get more intricate, introspecting the program and reason about what impact they may have at what point in execution becomes difficult. Teaching challenges in this module stall the program in critical areas, allowing students to experiment with carrying out their attack without time pressure (and resulting implementation difficulties in scripting fast attacks) before resuming the challenge execution at their leisure.

2.3.8 Kernel Security

The security of operating system kernels is considered to be a very advanced topic. Undergraduate OS courses rarely cover interactions with the kernel in any significant way, and undergraduate security courses rarely talk about the kernel at all (other than its effects of memory allocation and permissions). Our challenge design allows us to not only discuss the kernel, but instill in our students a deep-seated understanding of its security implications. This module has a fundamental effect on students in the sense that it expands their experience from “program” hacking to “system” hacking: the first time a student single-steps *into* (as opposed to *over*) a `syscall` instruction to find themselves debugging *the kernel syscall handler*, their perspective fundamentally expands beyond a typical undergraduate security course.

Teaching variants log outputs of kernel APIs used by the vulnerable code. This allows students to, for example, understand what data the kernel accesses from userspace applications.

2.4 Integration Challenge Design

To truly achieve proficiency, students must put the individual concepts learned in the modules above together into an end-to-end understanding of cybersecurity. We created two such integration modules: one focusing on the exploitation of individual *programs* (integrating Shellcode Injection through Memory Errors), and one focusing on the exploitation of multi-component *systems* (integrating *all* of the material). These challenges, despite their complexity, are combinations of prior modules. The teaching variant actions in those prior modules are preserved wherever possible.

2.4.1 Program Exploitation

As with the single-concept challenges, a core tenet of the progression is the gradual, bit-by-bit increase in complexity. Thus, the challenges start with a very simple buffer overflow, and become increasingly complicated from there. The final challenge of this module is a version of the VM from the Reverse Engineering module, reimagined as a realistic just-in-time compiler-based rather than interpreter-based VM. Students discover memory corruption issues within the virtual machine to hijack control flow of the program, then abuse the just-in-time compilation to introduce precisely-crafted shellcode. This sort of end to end exploit is conceptually similar to what modern web browser exploits against actual JavaScript engines look like in the real world.

2.4.2 System Exploitation

The second integration module extends into advanced material, exploring, e.g., the impacts of Race Conditions to enable Dynamic Allocator Misuse in multi-threaded software. These challenges convey two concepts: the chaining of radically different vulnerabilities to achieve a goal impossible with either vulnerability alone, and the importance of knowledge of the interconnectedness of program memory regions during exploitation.

These challenges integrate Kernel Security by reimagining the VM from the Reverse Engineering module as a kernel subsystem (similar to the BPF VM that powers the Linux kernel's network and system call firewalls). The final challenge (again, a series of challenges gradually leads to cumulative increases in complexity) requires students to perform a multi-stage exploit using concepts from all of the prior modules to first

race heap operations to achieve arbitrary code execution in a sandboxed userspace program, then pivot to attacking the virtual machine kernel subsystem by racing threads to induce memory corruption in the kernel, get a kernel pointer leak, and achieve arbitrary code execution in the kernel, all in order to finally disable the properly-configured seccomp sandbox (by rewriting relevant kernel data structures that track process sandboxing state), escalate privileges to root (again, by carefully corrupting kernel data structures), and get the flag.

2.5 Discussion

2.5.1 Smoothing Learning Curves

As challenges are responsible for progressively introducing concepts, it is critical to minimize jumps in difficulty throughout the progression of a module. We can measure difficulty jumps by analyzing discontinuities in challenge solve progressions in terms of the percentage of students solving each challenge. In our experience, when a large drop in solutions exists between one challenge and the next, it means that the more difficult challenge is conveying too many concurrent concepts. These concepts should be broken apart into one or more intermediate challenges which build up to that original difficult challenge.

A representative example of this evolution is our Reverse Engineering module. The Fall 2020 iteration of our course had a massive difficulty jump approximately 70% through the module, resulting in half as many solves as the previous challenge. This occurs when the first virtual machine reverse engineering challenge is introduced. Compared to the prior (much simpler) “crackme” challenges, students were required

to understand numerous new concepts concurrently in order to solve the challenge: obfuscation, interpretation of bytecode, and the use of complex data structures. Many students failed to make this jump. To remediate this, we added several conceptually simpler challenges in Fall 2021 to prepare students for this challenge. We did this by initially introducing pieces of the VM just as obfuscation for normal code logic, initially avoiding any sort of interpreter loop (and subsequent handling by the VM of its own control flow) or embedded VM instruction interpreters. Over the course of a number of challenges, we introduced these sub-concepts one by one, making it much easier for students new to reverse engineering to adapt. This resulted in a much smoother challenge progression and dramatically improved student success. As opposed to the 50% drop in solves of the first VM challenge, no challenge in Fall 2021 had a solve drop of more than 10%. At this point, we hit a limit in curve smoothness: in 2022, we added additional intermediate challenges because we felt that obfuscation concepts could be introduced even more slowly, but we did not observe a similar improvement in solution rates over Fall 2021.

2.5.2 Challenge Generation

We developed `pwnshop`, an open-source (available at <https://github.com/pwncollege/pwnshop>), template-based system for generating and maintaining our hundreds of individual challenges. This system leverages Jinja [40] as a templating layer on top of C source code. In doing so, we can conditionally include or exclude code, which is crucial for maintainably generating a challenge with just a small difference compared to a previous challenge. This capability enables us to fix a bug in a challenge (for example, disabling output buffering), and having that fix propagate to

all challenges that are based on that template. Furthermore, this allows us to generate challenges with randomized values, such as randomized passwords or randomized memory layouts. The benefits of this approach are twofold. First, it allows us to generate more versions of the same fundamental challenge, which can be useful for increasing student’s practice with a particular concept (for example, calculating buffer overflow offsets). Second, it allows us to generate challenges that are unique to each student, which can be useful for preventing students from sharing complete solutions with each other. In order to make sure that the challenges are still solvable, we have verify scripts that can solve any version of a challenge, and we use these scripts to verify that the challenges are solvable before they are released to students.

2.5.3 Refining Core CS Concepts

An interesting benefit of our challenges to students is the solidification of earlier Computer Science concepts that students may have previously missed. For example, despite many students having prior experience with process scheduling in our institution’s Operating Systems course, many of our students did not seem to truly “get” the concept until they had to exploit process scheduling intricacies to attack programs in the Race Condition module. We observed similar effects with concept missed in students’ Computer Organization courses, and with privilege and memory isolation in the Kernel Security module.

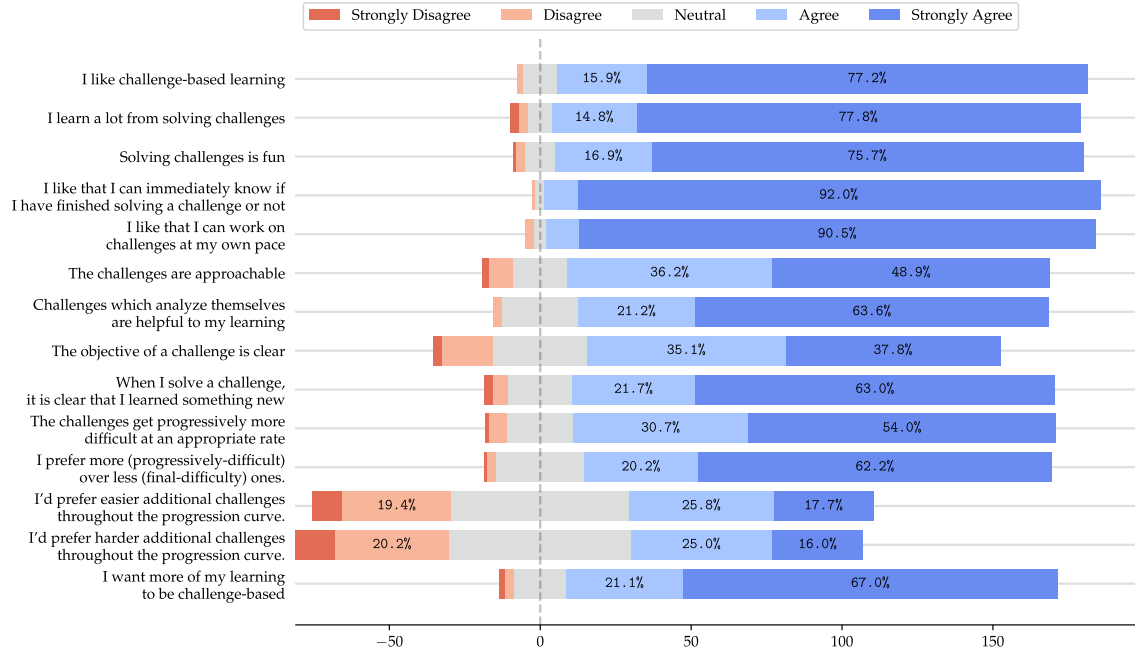


Figure 6. Survey Responses. Total number of participants to respond positively (postive values) and negatively (negative values).

2.6 Evaluation

2.6.1 Survey

To understand the effectiveness of challenge-based learning and our challenge design, we distributed an IRB-exempt (minimal harm) survey to our students. They survey contained both quantitative (Likert scale) and qualitative (free response) components. Since our challenges are openly and freely accessible to learners around the world, we received a wide variety of responses from 200 total respondents. In this section, we distill these responses into insights about the potential of our proposed learning approach.

We present our Likert question results in Figure 6. Our style of challenge-based

learning was popular: 93.1% of students reported liking challenge-based learning and 92.6% reported significant learning from these challenges. Students almost universally appreciated the immediate feedback. Our “teaching-variant” challenges were quite well-received, with 84.8% of students reporting being helped by them. Likewise, 84.7% of students learned something new from solving individual challenges and were satisfied with our challenges’ difficulty progression. In fact, 82.2% of students prefer *more* challenges, despite this course obtaining a reputation as the hardest course in our department. There does not appear to be consensus of whether these additional challenges should be on the easier or harder end of the progression. Overall, 88.1% of students would prefer more of their learning to be challenge-based. Questions about individual modules also confirmed the aforementioned refinement of Computer Science concepts.

Another theme was the duality of challenge frustration and the satisfaction derived from solving them. During course design, we often strive to avoid frustration (for example, by smoothing difficulty curves between challenges), but the framing of the responses suggests that some frustration might incentivize students to push through and “defeat” the challenge. In several modules, including Race Conditions, Kernel Security, Memory Errors, and Dynamic Allocator Misuse, students reported that carrying out attacks against related weaknesses helped them develop code resilient to such flaws. Further exploration of this phenomenon could be valuable to the Software Engineering field.

The qualitative responses shed light on some subtleties. While students find challenge-based learning rewarding and engaging, some note that the nature of the class can be “frustrating”, and that some challenges are repetitive or lack guidance. Even on these, students enjoy the practical application of skills and the immediate

feedback on their progress. Free form responses confirm that students appreciate the progressive difficulty within modules and the self-pacing that they enable. However, they suggest various offering lab time for hands-on, guided assistance with concepts to prevent languishing among the student body. Overall, they believe that challenge-based learning is an effective approach for cybersecurity education but there are clear paths for refined execution.

2.6.2 Follow-on Course

To assess how well the skills acquired from our challenge-based learning approach translated into practical applications, we created a follow-on course on applied vulnerability research. This course was offered to students who had successfully completed all challenges from our preliminary challenge-based learning course. The objective of this course was to give students the opportunity to find and responsibly disclose vulnerabilities in real-world open-source software. A total of 19 students enrolled in this course, forming 5 teams. The teams analyzed a variety of projects including PHP, MuJS, Pillow, Radare2, and a popular Gameboy Advanced emulator. All teams were able to find zero-day vulnerabilities in their respective projects, with several of those vulnerabilities translating into CVEs and a more secure global software ecosystem. Students focused on reverse engineering, automated vulnerability discovery (fuzzing), crash triaging, exploit development, and responsible disclosure.

While we haven't conducted a rigorous evaluation, it appears the skills from our challenge-based learning approach—especially in exploitation processes, reverse engineering, and modern exploit techniques—played a significant role in the course's success. There is a significant breadth and depth of knowledge and skills required

to even begin real-world vulnerability research; students must be able to understand threat models, and understand the technical implication of how different exploit primitives can be used together to exploit a vulnerability in order to truly compromise a system's security boundaries. We relied on the fact that all students in this course had, for example, already successfully abused dynamic allocators to leak information, bypass PIE, perform an arbitrary write, and achieve code execution through ROP—and had done so in several contexts, with different constraints, and spent hours debugging these exploit chains to completion. This allowed us to focus exclusively on the more advanced aspects of how to apply these skills to real-world software, with the aid of modern vulnerability research techniques. We are unaware of any other undergraduate university course that has been able to successfully teach these pre-requisite skills to students to the extent that we achieved.

Interestingly, we also discovered that while students were more than capable of producing exploits for the vulnerabilities they discovered, and several did, they were more interested in reporting the vulnerability without a full exploit, so that they could move on to finding more vulnerabilities. This in turn suggests that there are two critical tracks in vulnerability research: the ability to find vulnerabilities, and the ability to exploit them. While in this chapter we primarily focused on the latter and happened to achieve the former, further research is needed in order to better understand how this style of material can be optimized explicitly toward finding vulnerabilities in real-world software.

2.7 Conclusion

In this chapter, we described a design for a challenge-based cybersecurity curriculum that combines single-concept modules and integration modules comprised of challenges that gradually increase in complexity (conveying sub-concepts one by one) and help guide students through their own solutions. In running cybersecurity courses with these challenges, we observed not only great efficacy in student outcomes in cybersecurity, but a radical refinement of their understanding of core Computer Science concepts. To help the educational community, we have made these challenges, as well as the accompanying lecture material, freely available to the world.

Chapter 3

DOJO: APPLIED CYBERSECURITY EDUCATION IN THE BROWSER

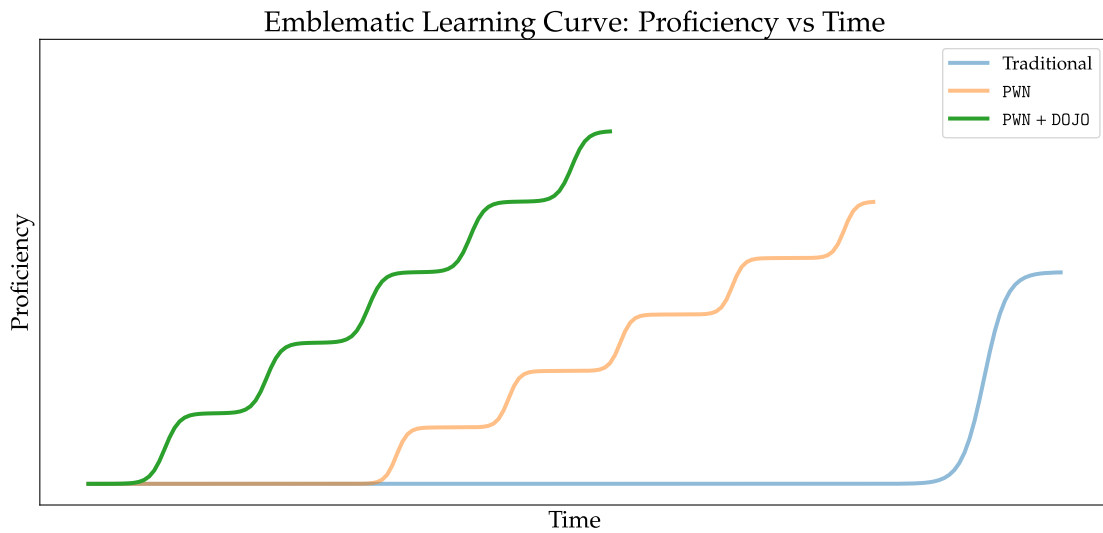


Figure 7. Emblematic Learning Curve: PWN + DOJO

The DOJO infrastructure is designed to support the PWN methodology, and in doing so, shortens the learning curve for cybersecurity education. Students are able to *immediately* begin working on cybersecurity challenges to learn cybersecurity concepts, entirely within the browser, without the need for any additional software or hardware. Teachers are able to *immediately* deploy new challenges and other educational materials, which are immediately available to all students. In doing so, we are able to significantly reduce the barrier to entry for cybersecurity education.

3.1 Introduction

In the rapidly evolving landscape of software development, containerization has been instrumental in ensuring portability and reproducibility of production, testing, and more recently, development environments. By bundling all of the dependencies required for a system within a single portable container, developers can effortlessly set up an identical version of a live working system, implement modifications, validate them, and deploy to production, ensuring a consistent behavior across different environments.

The profound impact of this approach is not limited to production and testing. Developers are now leveraging containerized development environments to streamline and standardize their workflows [34]. This paradigm offers immediate access to a fully configured development environment, with all essential tools, from compilers and linters to debuggers and profilers, pre-installed and set to go. Such an environment mitigates the challenges of individualized setups and fosters team consistency, especially aiding novice developers who might be daunted by the intricate setup processes.

GitHub Codespaces, a cloud-based service, exemplifies this trend by offering developers the capability to set up their development environment, filled with necessary tools, on a robust machine that can be accessed universally [33]. This not only expedites the code development and testing process but also enables quick onboarding, particularly useful when working with various devices or when switching to a new one.

A parallel can be drawn in the realm of applied education, where such containerized environments are being integrated [52, 53]. Novice learners, akin to new developers, often face challenges in setting up their learning environments, which may serve as deterrents to their educational pursuits. Addressing these barriers becomes paramount,

especially in fields like cybersecurity, where the demand for trained professionals outstrips the supply due to a lack of hands-on experience and effective training methodologies [23, 39].

Given the pressing need for effective learning environments and drawing inspiration from the containerization wave in software development, we introduce DOJO, a platform designed specifically to facilitate hands-on cybersecurity education. DOJO adopts principles from the realm of Capture The Flag competitions [24], wherein learners are tasked with solving challenges, thereby obtaining flags as evidence of their newfound skills. Instead of imposing the onus of environment setup on learners, DOJO offers a pre-configured environment, available through both browsers and SSH, allowing students to dive into hands-on cybersecurity challenges instantly. In contrast to existing platforms, **our primary focus is to allow students to execute *every step*—discovery, implementation, and debugging—of even the most advanced and technical challenges directly within the DOJO environment.**

DOJO is freely available for use at <https://pwn.college/> and is open-source at <https://github.com/pwncollege/dojo>. Educators can seamlessly launch their own version of DOJO on their own hardware with a simple `docker run` or establish a *private dojo* on our hosted platform using just a `git push`, incorporating their own challenges or leveraging our existing ones. Over the last 5 years, DOJO **has been used by more than 10,000 students** from over 100 countries and over 100 academic institutions. We know of at least 10 courses around the world that have used DOJO as a critical component of their curriculum. In total, DOJO has facilitated more than 1,000,000 challenge solves, and has attracted thousands of learners with no course requirement or incentive other than the desire to learn.

In this chapter, we explore the architecture and ethos behind DOJO, specifically

focusing on the infrastructure itself rather than the educational content that may be hosted within it. Our insights, shaped by its creation and enriched by our personal interactions with it, are further enhanced by direct feedback from students. This offers readers an authentic look into students’ experience with using DOJO. While our lens is primarily on applied cybersecurity education, the insights shared may have broader relevance.

3.2 Related Work

There are various projects that address the challenge of providing students with access to a learning environment that enables them to work on hands-on challenge problems and experiment with cybersecurity concepts. This section delves into the relevant existing works and compares them with our contribution. Moreover, we have provided a summary of the significant features of our work, in relation to notable projects in the field, in Table 1.

3.2.1 Capture The Flag Projects

Capture The Flag (CTF) is a cybersecurity competition style in which *flags* (represented by secret data stored in privileged areas of a system) must be *captured* (disclosed and redeemed for credit) by participants through compromising the security of some system. Recently, CTF has been increasingly used in cybersecurity education [5, 28, 57, 14, 63].

Among CTF projects, picoCTF [15] is the most similar to DOJO. It offers a challenge frontend (to view what challenges are available), backend (to run challenges), and

working environment (to work on challenges). A key difference from DOJO is that the challenge environment and working environment are separate. The provided working environment grants users remote capabilities when interacting with the challenge (e.g., over `netcat`), reducing introspection capabilities and hampering student abilities to reason about what is happening. Additionally, the working environment is only accessible through a simple in-browser terminal (a solution which DOJO once used, but has since iterated and improved on), not SSH (advanced users may prefer), VS Code (for editing code), or a desktop environment (for graphical interface tools). Furthermore, the project is no longer under active open-source development, although a public version of the infrastructure is available and maintained, especially for the yearly picoCTF competition.

"Its cool that DOJO provides an environment for solving challenges and without it, I would have to set up my own environment (creating a VM, installing necessary packages, fighting to getting correct dependencies and so on). This is where picoCTF falls behind (although its also a good website for CTF)." Student Feedback

CTFd [16] is a very popular frontend for running CTFs. However, all though some features for running challenges are available, it is not the primary focus of the open-source project. Furthermore, a working environment is not provided. Root The Box [10] and fbCTF [30] are other popular frontends. kCTF [35] is a popular backend for running challenges on Kubernetes, but does not provide a frontend. iCTF [90] allows users to run an attack-defense CTF, in which participants exploit each other participant's vulnerable services, and likewise defend against attack.

3.2.2 Wargame Platforms

Wargames are collections of sometimes-educational challenges available for participants to tackle. OverTheWire [97] and pwnable.kr [74] offers participants the ability to access a number of challenges through SSH. pwnable.tw [75] and archive.ooo [69] only allow remote interactions (e.g., over `netcat`), and do not provide a working environment. While the former two offer some visibility into the challenge environment, the latter two are modeled strictly from the perspective an external actor. An understanding of the environment is only made possible through this remote interaction, and file downloads of the underlying programs. In the case of all four of these projects, and in contrast to DOJO, introspection and tooling are not a focus: a local working environment is necessary.

3.2.3 Cyber Range Projects

Cyber Ranges are designed as environments for exploiting software in a manner that more closely mirrors real-world scenarios. They often target known vulnerabilities in actual software, making them a more realistic training ground compared to the more game-like nature of CTFs.

KYPO [92] most closely aligns with DOJO as it combines the challenge and work environments. Students move from an initial scenario accessible via SSH or an in-browser desktop to larger challenge environments. However, KYPO only supports features required by the challenges themselves, and excludes critical debugging and troubleshooting functionality. This design reflects the constraints of real-world situations, but hampers the ability for confused students to fix their understanding.

| Project | Layer | Openness | Interaction | Isolation | Environment |
|-------------------|--|---|---|--|---|
| | Frontend Backend Working Environment | Open Source Open Deployment Free To Use | Download Netcat Website In-Environment | Isolated Instances Dynamic Challenges | SSH In-Browser Terminal In-Browser Code Editor Persistent Desktop Rich Tooling Privileged Access |
| CTFd | ● ● - | ● ● ● | ● ● ● - | - - | - - - - - |
| CTFd Enterprise | ● ● ● | ● ● - | ● ● ● - | ● - | - ● - - - |
| picoCTF | ● ● ● | ● ● ● | ● ● ● - | ● ● | - ● - - ● - |
| fbCTF | ● - - | ● - ● | ● ● ● | - - | - - - - - |
| Root The Box | ● - - | ● ● ● | ● ● ● - | - - | - - - - - |
| kCTF | - ● - | ● - ● | ● ● ● - | ● - | - - - - - |
| iCTF | ● ● ● | ● ● ● | ● ● ● ● | ● - | ● - - - ● - ● |
| OverTheWire | ● ● ● | - ● ● | ● ● ● ● | - - | ● - - - - |
| pwnable.kr | ● ● ● | - ● ● | ● - - ● | - - | ● - - - - |
| pwnable.tw | ● ● - | - ● ● | ● ● - - | - - | - - - - - |
| archive.ooo | ● ● - | ● ● ● | ● ● ● - | ● - | - - - - - |
| SecDevOps@Cuse | - ● ● | ● - ● | ● ● ● ● | ● - | ● - - - - ● - |
| Alpaca | - ● ● | ● - ● | ● ● ● - | ● ● | - - - - - |
| SecGen | - ● ● | ● - ● | ● ● ● - | ● ● | - - - - - |
| CyTrONE | ● ● ● | ● - ● | ● ● ● ● | ● - | ● - - - - ● - |
| KYPO | ● ● ● | ● - ● | ● ● ● ● | ● - | ● - - ● - ● - |
| TryHackMe | ● ● ● | - ● ● | ● ● ● ● | ● - | ● ● - ● - ● ● |
| TryHackMe Premium | ● ● ● | - ● - | ● ● ● ● | ● - | ● ● - ● - ● ● |
| HackTheBox | ● ● ● | - ● ● | ● ● ● - | - - | ● - - ● ● ● ● |
| HackTheBox VIP | ● ● ● | - ● - | ● ● ● - | ● - | ● - - ● ● ● ● |
| RET2 Wargames | ● ● ● | - ● ● | - - - ● | ● - | - - ● ● - ● - |
| DOJO | ● ● ● | ● ● ● | ● ● ● ● | ● ● | ● ● ● ● ● ● ● |

● = provides property; ● = partially provides property; - = does not provide property;

Table 1. Features of Cybersecurity Education Projects. The projects are grouped into, in order, CTF Platforms, Wargames, CyberRange Projects, and Commercial Training Platforms, as described in Section 3.2.

In contrast, DOJO optimizes for transparency and debuggability within the challenge environment, giving students critical tools to maximize understanding. Additionally, KYPO’s infrastructure is resource-intensive: catering to a large student cohort is challenging because KYPO relies on individual virtual machines for every student instead of DOJO’s more efficient container approach [73]. For scale, the KYPO project

recommends that students run virtual machines locally, forgoing the benefits of a turnkey, centralized environment.

CyTrONE [8] is a challenge hosting backend, but it lacks a centralized working environment and even a challenge frontend (instead, it uses the Moodle Learning Management System [60]). SecDevOps@Cuse [81] places some emphasis on including security tools in its working environment, but does so without a challenge frontend. Projects like Alpaca [29] and SecGen [80] craft challenge environments dynamically from vulnerability datasets, but do not focus on students' working environments.

3.2.4 Commercial Training Platforms

TryHackMe [88] has both a challenge environment, and a working environment. Challenge environments vary dramatically from challenge to challenge. For some challenges, the challenge environment serves as the working environment. However, in cases where the challenge environment is either remote (e.g., some service is only accessible over the network), or tooling is insufficient within the local challenge environment, an additional dedicated working environment is provided. Within the working environment, students have access to a standardized set of tools and privileged access to further customize the environment. The dedicated working environment may only be accessed from an in-browser desktop. HackTheBox [9] has separate challenge environments and working environments. The working environment contains a standardized set of tools, with the ability to persist a student's working data across challenges, and privileged access to further customize the environment. However, the working environment *is not* the challenge environment. It is an environment from which students can *work* on the challenge, with the same effective capabilities

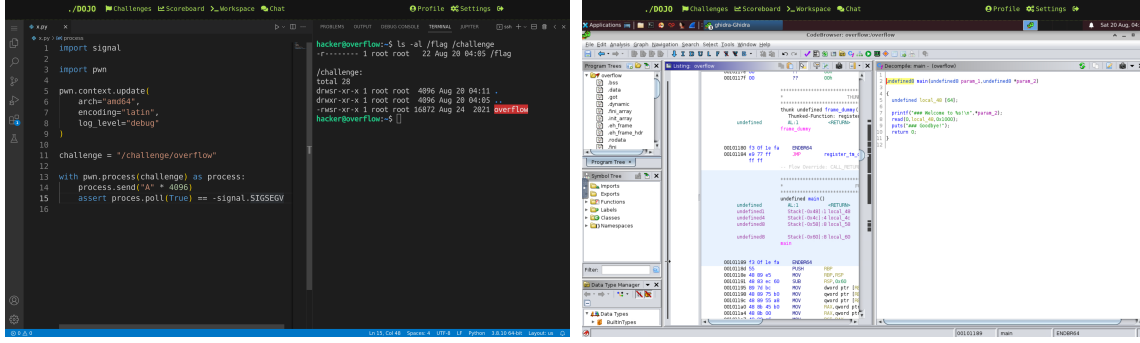


Figure 9. DOJO’s browser-based workspace. Left: VS Code, text editor alongside a terminal. Right: GUI reverse engineering tool.

that local working environments have. This prevents full challenge introspection and debugging, as enabled by DOJO. RET2 Wargames [78] has a tightly integrated challenge and working environment. Students work in a web environment which *simulates* standard tools in order to solve the challenge. However, arbitrary tools are not available, and because tools are simulated, the transferability of learned knowledge beyond the simulated environment may be hampered.

3.3 Dojo Design

Our priority in designing DOJO was to maximize the ease of challenge access for students, and of challenge deployment for instructors. In doing so, we leveraged a number of existing technologies, focusing on smooth integration. DOJO takes inspiration from developments in cybersecurity competition, specifically from Capture The Flag competitions [24], and expands them to accomplish our goal of democratizing cybersecurity education access.

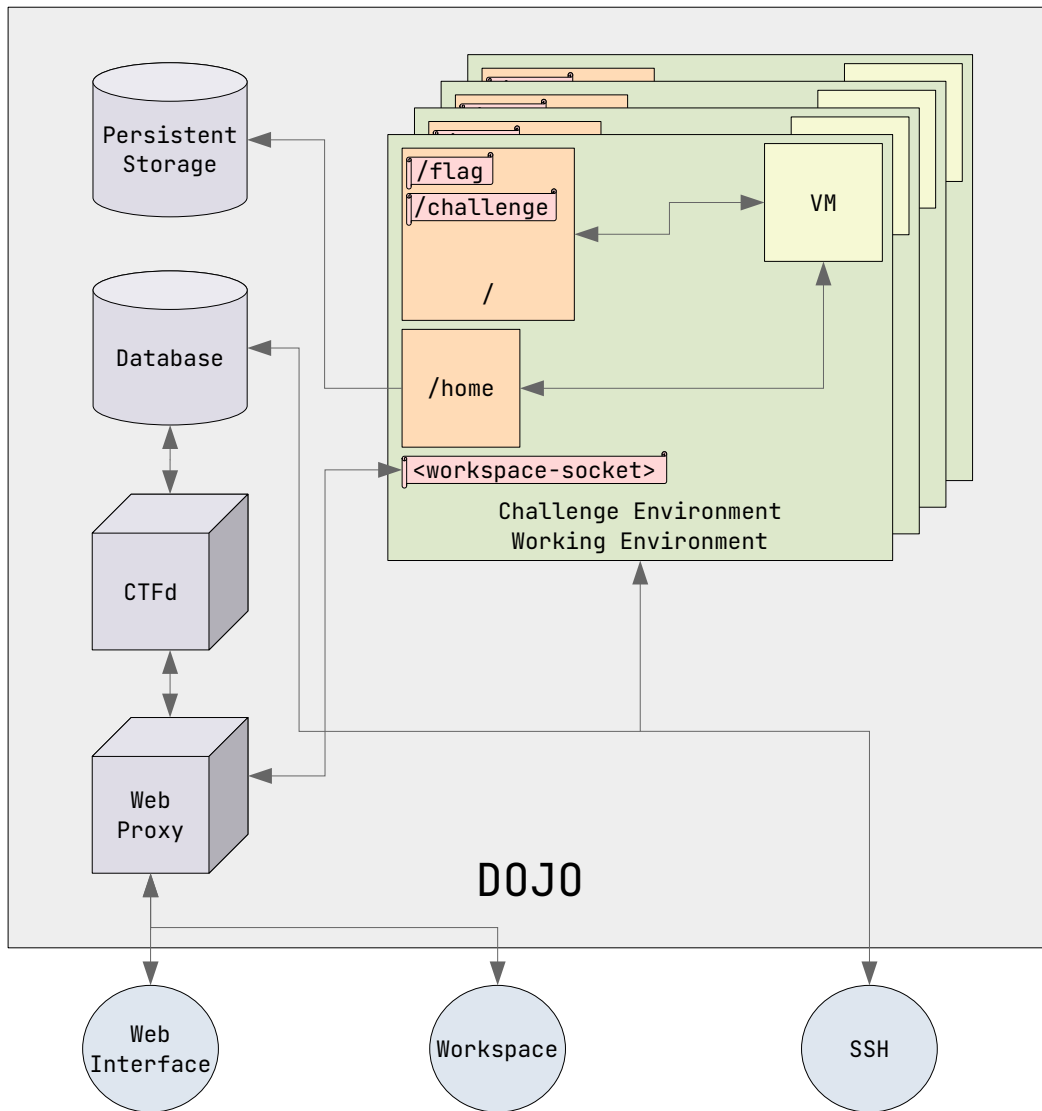


Figure 8. The overall design of DOJO. Student environments are isolated in containers, managed by a series of other components. For coursework that requires it, DOJO can run nested virtual machines (for example, with vulnerable kernel configurations).

3.3.1 Challenge Environments

DOJO heavily extends the CTFd framework, which is popular for facilitating simple one-off “Jeopardy-style” CTFs [16]. CTFd manages user accounts, tracks flag submissions, and provides a basic web interface which lists the challenges and displays a scoreboard. DOJO builds upon CTFd to provide a long-term comprehensive environment for students.

In standard CTFs, participants are given a downloadable challenge program or instructions for how to communicate with a remote running challenge (for example, `nc <IP> <PORT>`). We have expanded upon this by providing students with a dedicated containerized environment for each challenge, which they can *start* when they are ready to work on it. Once started, the container contains the challenge and any other necessary files in its filesystem.

Inspired by CTFs, DOJO stores a `/flag` file in each container, which students access by solving the challenge and then submit to DOJO to count the solve. The `/flag` file is only accessible to challenge programs, which have SUID permissions to enable this access. DOJO ensures that no other files are SUID, reducing the risk of inadvertent flag retrieval due to environment misconfiguration. This method highlights the advantages of a centralized educational platform over local setups where students might have unrestricted access. Further, it immediately conveys to students when they have completed the challenge.

"DOJO removes a lot of the frustration involved with "it compiles on my machine, but not the grader's"."

Student Feedback

3.3.2 Working Environments

3.3.2.1 Extensive Tooling

Students work on challenges in the same environment that hosts the challenge. These containers come pre-configured with essential security tools. Currently the default DOJO challenge environment comes with `ipython`, `tmux`, `strace`, `gdb`, `pwntools`, `pwndbg`, `gef`, `radare2`, `ghidra`, `wireshark`, `nmap`, `scapy`, `requests`, `curl`, and many other tools. The goal is to allow students to execute every step of the challenge—discovery, implementation, and debugging—directly within the DOJO environment.

"Hacking very often comes down to the most nitty gritty of details, and often you need specific tooling to save time. Giving someone a set environment where all of those details are controlled, and where they know they have all the tools they need, is a godsend for learning."

Student Feedback

This setup ensures consistent environments for students and allows new tools to be added (for example, to support new types of challenges) by modifying the `Dockerfile` of the challenge environment, which makes the change immediately available to all students. This spares students from potential frustration in tooling setup and eliminates time instructors might spend assisting students with installation problems.

3.3.2.2 Privileged Mode

DOJO also offers a *privileged* mode when starting the challenge. In this mode, students work in an environment similar to the standard mode, but with additional `sudo` privileges and a placeholder flag. With this root access, students can fully

analyze and debug a challenge program, which is otherwise not possible in some circumstances. For example, if challenge behavior is strictly dependent on being able to access the flag (for example, to encrypt it), then any debugging of that challenge must be done with flag access capabilities. This mode enables this without otherwise leaking the actual solution flag until the student re-runs their final solution in the standard mode, ensuring the integrity of the challenge.

3.3.2.3 Persistent Home Directory

In order to improve usability when switching between challenges and challenge modes, DOJO persists the student's home directory across all challenge environments. This persistence is essential as it allows students to store previous solutions (for reference and expansion), as well as custom scripts and tools. Critically, the home directory is mounted with the `nosuid` option, which prevents students from smuggling a root-owned SUID binary across different environments.

"I really like the infrastructure, and I much appreciate that you have permanent storage! Even though I have most things set up locally, I still work in DOJO most of the time."

Student Feedback

3.3.3 Environment Access

3.3.3.1 SSH

Students may access their challenge environment after starting a challenge in one of three ways. The first way is through SSH. Students may upload a public key within the DOJO web-interface, allowing them to SSH into DOJO and transfer files with `scp`.

"The fact that I can SSH into DOJO during lunch breaks makes this platform stand out to me."

Student Feedback

3.3.3.2 VS Code

The second way is through an Visual Studio Code running in the DOJO and exposed through the student's web browser [18], as seen in Figure 9. VS Code provides a powerful text editor, command line terminal, file explorer (including file upload and download), and a plugin interface with many community-created plugins.

"One of the biggest reasons I use VS Code to interact is because my device becomes slow if I run a VM."

Student Feedback

3.3.3.3 In-Browser Desktop Environment

The third way is through an *In-Browser Desktop Environment*, as can be seen in Figure 9. This provides students with the ability to run arbitrary graphical user interface programs (which is relevant in advanced security curricula, such as Software

Reverse Engineering) from within the browser. This method makes it possible for students to learn *entirely* through a student's browser, enabling full DOJO access not only from laptops, but also tablets or other mobile devices, entirely removing software or hardware barriers to entry from the student's perspective. Education is as simple as starting a challenge, and then immediately having access to a full computing environment, all from within the browser.

"I can't stress enough how helpful DOJO was. Being able to access DOJO from anywhere (home, work, etc.) meant I could pick up where I left off." Student Feedback

3.3.4 Challenge Virtualization

While `docker` works for a large number of challenge use-cases, it may not always provide enough capabilities for some challenges. For example, a standard unprivileged container cannot arbitrarily manage networking resources, hampering education in network security. DOJO has two answers to these limitations.

3.3.4.1 User Namespaces

DOJO supports nested namespace virtualization, enabling challenges to create arbitrary networking topologies, process isolation, and permission models (among other namespace features). This requires DOJO to modify `docker`'s `seccomp` filter to allow relevant system calls (e.g., `unshare`) and to create a *user namespace* to grant necessary Linux `capabilities` to the challenge without compromising the security of the overall DOJO system.

While this is *intended* to be secure according to the namespace security model, unprivileged user namespaces have previously caused serious kernel vulnerabilities. As the implementation of this Linux subsystem continues to mature, more vendors have been willing to trust this configuration, and it seems likely that these features may be supported by default in a future version of `docker`.

3.3.4.2 Virtual Machines

Nested namespace virtualization does not support *all* challenge use-cases. Namespaced processes operate within a single shared kernel, which cannot securely support challenges including, for example, vulnerable Linux kernel modules. Furthermore, some scenarios might demand the operation of an entirely different kernel—for example, to run varying versions of Linux, or even a completely different OS like Windows or MacOS. To overcome these limitations, DOJO supports both virtualization and system emulation *within* the user container.

Critically, our virtual machine implementation transparently shares the container’s file system. This means that all files are automatically shared between the container and virtual machine, and that modifications in one environment are immediately reflected in the other. This includes the persistent home directory (see Section 3.3.2.3). This feature eliminates hurdles of transferring solution and debug code into traditionally-minimal environments running in vulnerable kernels, which has traditionally frustrated students tackling such problems. To our knowledge, DOJO is the first cybersecurity environment with this capability.

DOJO uses QEMU [7], with `kvm` providing near-native virtualization performance, `9p` supporting the transparent filesystem mapping, and `SLIRP` mapping networking

between the virtual machine and container. Connecting to the virtual machine happens transparently over SSH. Debugging mode automatically connects to the QEMU gdb stub and loads debug symbols from the virtualized kernel.

"For the kernel challenges, the DOJO infrastructure was very helpful. I could probably have figured out how set up QEMU and stuff locally, but it would have taken time and effort and I don't think I would have gotten to it from never having tried anything kernel before."

Student Feedback

3.3.5 Instructor Capabilities

3.3.5.1 Environment Sharing

One unique capability is bidirectional information sharing. Instructors can broadcast their desktop environment (described in Section 3.3.3) to students, allowing them to view what the instructor is doing live during lectures and demonstrations. In the other direction, instructors have access to interacting with students' desktop environments (viewing *and controlling*) or alternatively access their environment through SSH. Through this, an instructor can understand directly how a student is approaching a problem, both real-time and after-the-fact by examining their solution scripts. This capability makes it much easier for instructors to remotely assist students with complex problems, which can be especially useful in a hybrid or online course. Of course, this instructor access is not a surprise to students: students are informed of these capabilities at the beginning of the course.

3.3.5.2 Anti-Cheat

DOJO offers several generic anti-cheat mechanisms. To thwart the sharing of flags, challenge flags are cryptographically generated for each challenge and user. This allows DOJO to verify that a flag is correct for a specific user and challenge, and to automatically detect flag sharing between students. To complicate the sharing of solutions among students, rather than giving all students the exact same challenges, instructors can specify multiple slight variations of the same challenge, with each student being randomly assigned one variation. This requires students to develop a solution which specifically solves their challenge.

3.3.5.3 Automatic Grading

In order to enable instructors to run courses with large numbers of students, DOJO supports automatic grading. The DOJO exports detailed statistics on student progress through challenges that instructors can use both for determining grades and to understand how long students spend working on each challenge. Coupled with the ability to view student desktops, and access student files, this allows instructors to easily understand how the class is progressing, to identify students who are struggling, and to identify students who are cheating.

3.4 Discussion

We have run courses (and portions of courses) at our university using DOJO for several years, through various stages of feature refinements, in various courses (ranging

from introductory to advanced). In this section, we discuss our observations and experiences, in the hope that they are useful for future educators.

3.4.1 Deploying DOJO

DOJO is designed to be modified and deployed to support the greater educational community. The infrastructure is contained within a single docker container that uses a single directory on the host for all data storage and facilitates all DOJO communication over ports 22, 80, and 443 for SSH, HTTP, and HTTPS. Once the image is built, and container started, it automatically deploys all dependencies, generates a valid HTTPS certificate, loads the database with instructor-provided challenges, and generates persistent user home directories. Everything happens automatically. Each component is isolated for easy customization, and core logic is engineered so that others may make changes as they see fit.

Turnkey deployment has enabled DOJO's use by our colleagues to facilitate several courses at our institution and has been used by several educators around the world.

3.4.2 Supported Challenge Types

We have used DOJO to host challenges covering topics ranging from basic Linux usage and shell scripting to network security, web security, cryptography, reverse engineering, memory corruption, kernel security, and even microarchitecture speculative execution attacks. One challenge type that we found inconvenient to support by DOJO are things involving inter-student interactions, such as group-based GPG code-signing exercises. This is a current focus of DOJO improvement for us.

3.4.3 An Open Deployment

As an alternative to running their own DOJO instance, instructors can create a *private dojo* inside our running DOJO instance with an arbitrary set of challenges. This enables instructors to run security classes, using our cutting-edge infrastructure, with no computing resources and nearly no setup. We know of eight universities in five countries that use our centrally-hosted DOJO to power their actual cybersecurity courses, and the infrastructure has become a standard way for student clubs to teach their members about cybersecurity. This seamless usage of DOJO speaks to its potential impact in the field of cybersecurity education.

3.4.4 Rapid Learning Launch

DOJO has facilitated accelerated cybersecurity workshops in settings where students only had access to low-resource and non-privileged laptops. Having access to an environment in which everything could be done in the cloud proved to be invaluable for this experience. Students were able to show up and *immediately* dive straight into the challenges without wasting already-limited time to get basic tooling installed, or troubleshooting installation errors, or requiring exemptions to security policies. Additionally, the ability to share DOJO screens through the browser facilitated demonstrations without requiring access to a projector.

3.4.5 Scaling and Compute Requirements

D0J0’s use of containers (rather than full VMs) for most challenges allows for optimal usage of resources [42]. We currently run the central instance of the D0J0 infrastructure on a server, generously provided on a long-term basis by our institution, with 40 cores, 256 GB of memory, and 8 TB of storage. In our experience, this has been sufficient to run D0J0 for over 10,000 registered users, with ample computing resources still available. In fact, we have observed D0J0 accommodating 250 concurrent students without noticeable performance impact on individual users. The important insight for understanding how so many users can concurrently utilize the system is that most time is spent idle, as students contemplate ideas, in lightweight containers.

3.4.6 Impact of Feature Improvements

Over the years, we have observed the impact of improvements to D0J0 on the reduction of hurdles and frustration for students (especially novice ones).

One example is the workspace, which evolved significantly over the course of five years. This started out as a `netcat`-based interface without persistent storage or a GUI, resulting in significant load for students to configure local analysis environments. Our next iteration provided an `SSH`-based environment and persistent home directories, but the high student count at our institution resulted in a high instructor load even helping students configure their `SSH` clients and helping them install GUI-requiring security tools locally. Next, we deployed a fork of Google Chrome’s in-browser terminal to eliminate the requirement for students to set up `SSH`, but still faced student and instructor frustrations with local installation of GUI-requiring security tools and `SCP`

clients for file transfer. In the next iteration, we addressed the **SCP** issue by upgrading the terminal to a **DOJO**-hosted version of VS Code, which supports file transfer. Finally, we added the in-browser, VNC-based desktop to provide centralized access to GUI security tools. Each of these advancements reduced the amount of student frustration (and may help explain a decrease in our security course drop rate) and instructor support load for student tooling setup, the latter now reduced to near zero.

DOJO improvements have been even more keenly felt in the kernel security space, which is described in Section 3.3.4.2. The initial **DOJO** VM support followed the typical model of kernel security challenges in CTFs: a minimal VM with a vulnerable kernel and an embedded userspace. We used this model for two years of kernel security education, and found that students were actively discouraged from tackling this material because of the difficulty of challenge interaction in this model, rather than the difficulty of the challenges themselves. When we implemented the sharing of the host filesystem between the challenge container and the vulnerable VM, student frustration with interaction issues dropped to near-zero.

3.5 Evaluation

We conducted an IRB-exempt (minimal harm) survey to evaluate the impact of the latest version of **DOJO** on student learning experience. We received 200 responses from **DOJO** users around the world (including, but not limited to, students at our institution). Feedback excerpts are replicated throughout the chapter, and results of the quantitative portion of the survey are summarized in Figure 10, clearly showing that students overwhelmingly prefer the **DOJO** over even their own local environments.

We also analyzed the data according to gender. Of gender reporting respondents,

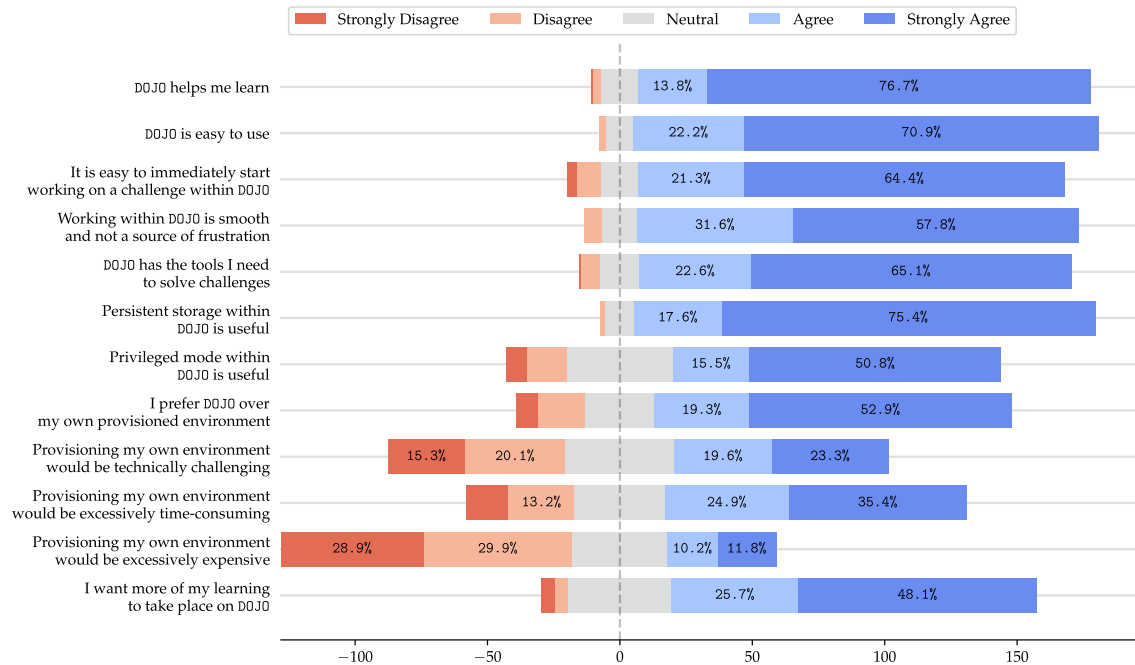


Figure 10. Survey Responses to 5-point Likert scale questions from 200 students about their experience with DOJO.

89% identified as Male, 9% as Female, and 2% as other responses. DOJO is more overwhelmingly preferred by non-Male students than by Male students. While 70.5% of Male students prefer using the DOJO, 94.4% of Female students and 100% of non-binary/other students do so. Similarly, a higher portion of non-Male students (55.6% of Female students and 75% of Other students, compared to 41% of Male students) had more concerns with the technical challenge of provisioning their own environments. In a field struggling with gender inclusivity, we feel that any leveling of the playing field is important, and hope that the DOJO can be a part of the puzzle of increasing the gender balance in Cybersecurity.

3.6 Conclusion

We developed **DOJO**, a state-of-the-art, open-source learning platform for hands-on cybersecurity education that aims to minimize barriers for both students and instructors. We described its features, delved into technical implementation details, discussed implications and our experience, and presented a survey of student experiences. **DOJO** has been successfully used in multiple university courses and workshops, is open source, and is available for free to the world. More than 10,000 students from around the world have already benefited from using **DOJO**, and we hope that this is just the beginning.

SENSAI: LARGE LANGUAGE MODELS AS APPLIED CYBERSECURITY TUTORS

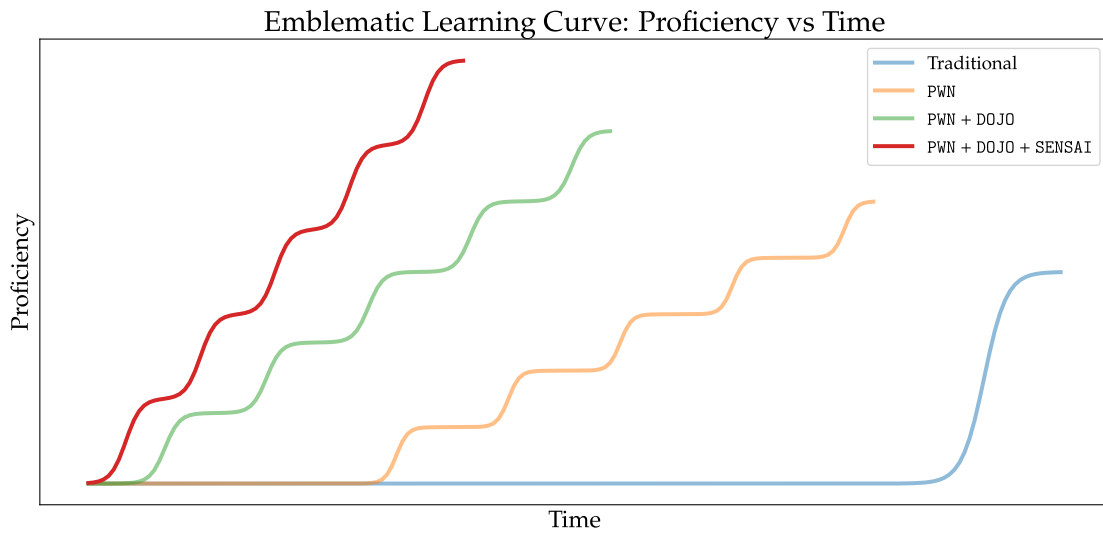


Figure 11. Emblematic Learning Curve: PWN + DOJO + SENSAT

The SENSAT tutor is enabled by the PWN methodology and the DOJO infrastructure, and is able to further improve the learning curve for cybersecurity education. Students are able to seek help from SENSAT, which is able to “look over their shoulder” and provide automated, immediate, individualized feedback and guidance. In doing so, we are able to provide a more individualized learning experience, at scale.

4.1 Introduction

What should a student do when faced with a seemingly insurmountable challenge?

The answer is shockingly simple: ask for help. Receiving guidance from a tutor has been a cornerstone of education for over a millennia—from Aristotle, who was readily available to guide his pupil Alexander the Great, to today’s modern classroom dynamics.

However, the modern educational landscape presents a unique challenge. As education expands, it is difficult to maintain the tradition of personalized mentorship; with large class sizes, a single instructor might be responsible for hundreds or even thousands of students. While current research is split on the exact impact of class size on educational outcomes [1], particularly in higher education, it is clear that more students in a class decreases the time that a professor can devote to assisting any particular student.

Teaching assistants can extend the reach of personalized mentorship and assistance, but even with this additional layer of support, the student-to-educator ratio is often significant. A lack of individualized assistance can be a significant barrier to learning. If a student is stuck on a problem, they may not know how to proceed. *Who should they ask for help?*

Peer-to-peer interactions have emerged as a potentially scalable solution, as the capacity for peer tutoring grows with the student population. These methods offer opportunities for students to exchange ideas, troubleshoot, and learn collaboratively—leading to improved academic performance [100, 12]. However, there are clear drawbacks: peers may lack the expertise and pedagogical skills of seasoned educators, which can limit the depth and effectiveness of their guidance [43].

Cybersecurity requires a deep understanding of many complex underlying computing concepts as well as the ability to precisely apply these concepts in practice. A lack of hands-on experience has been identified as a significant barrier to meeting the demand for trained cybersecurity professionals [23, 39]. This lack of trained cybersecurity professionals, in turn, leads to a lack of qualified cybersecurity educators, which further exacerbates the problem of a lack of hands-on experience, and so, a lack of trained cybersecurity professionals—a cycle that is difficult to overcome.

The advent of Large Language Models (LLMs) presents an exciting development in the quest to retain individualized educational assistance. Capable of seemingly understanding and processing a vast range of linguistic inputs, a key question is if these models can bring us a step closer to a truly intelligent automated tutor that can adapt its responses to the specific needs and context of a student. A key benefit is scalability, as an increase in the number of students does not diminish the ability to provide focused, individual attention. Furthermore, as the behavior of these models can be directed towards an educational mission, they offer a solution to the limitations associated with peer-to-peer interactions.

Unfortunately the capabilities of LLMs in this space are not yet fully understood. Furthermore, off-the-shelf LLMs lack the ability to “look over the learner’s shoulder”, to see how they are approaching a problem. This is important: learners often do not understand the problem well enough to ask the right question—*they don’t know what they don’t know* [84]. This is where a human tutor can be invaluable, as they can observe the student’s approach, intuit their misunderstandings, and provide targeted assistance.

In this chapter, we present **SENSAI**, an AI-powered tutoring system that leverages LLMs to provide personalized feedback and assistance to students. Crucially, **SENSAI**

features a novel design that automatically extracts the learner’s context—their recently active terminals and edited files—when they pose a question. This allows **SENSAI** to “look over the learner’s shoulder”, see what the learner is working on, and use this specific nuanced context to provide personalized and direct assistance. We deploy **SENSAI** across an applied cybersecurity curriculum at a large public R1 university, as well as to a broader online community of global learners. We discuss the design and implementation of **SENSAI** and further investigate its capabilities and impact. To gain a deeper understanding of its influence on the learning experience, we manually review over 500 **SENSAI** sessions and draw insights from these interactions, quantitatively measure the impact of **SENSAI**, and survey learners to understand their experience.

Over the last six months, **SENSAI** has been used by 852 learners to assist with 504 educational challenges. This has led to an exchange of 34,108 messages across 3,583 sessions, for a total cost of \$1,515—comparable to the cost of a single undergraduate teaching assistant, but with a much larger reach. We show that access to terminal and file context is *crucial* for **SENSAI**’s performance, noting a median solve time after interacting with **SENSAI** of 3 minutes, 46 seconds for instances where **SENSAI** had access to consistently “active” learner context, compared to 43 minutes, 48 seconds when **SENSAI** did not have any access to the learner’s context—a more than 10x difference. We further find that **SENSAI** is particularly effective at providing guidance on Python and Linux commands, but less effective at providing guidance on Reverse Engineering and GDB, where the context is more difficult to succinctly extract—limitations of the context window—and the topics require a deeper understanding and methodology to be effective—where the LLM’s “reflexive intelligence” is not as applicable. Overall, we find that **SENSAI** is a useful tool that enhances the learning experience and shifts the ratio of students to educators back in favor of the students.

4.2 Background and Related Work

4.2.1 Large Language Models

At the heart of Large Language Models (LLMs) is a remarkably simple, yet powerful concept: predicting the next token based on a sequence of preceding tokens, a probabilistic process recursively repeated to generate (hopefully) coherent text. A token is a single unit of text, such as a word or a punctuation mark. This prediction task is trained over substantial volumes of text data, enabling the model to learn patterns, contextual relationships, ideas, and rules of the language.

LLMs operate on a fixed-length context window, which is a sequence of tokens that the model considers when making predictions. It's important to note that this window spans both the input *prompt* tokens and the output *sampled* tokens. During its recursive process of token generation, the model focuses its attention [89] on this combined pool of tokens within the window to sample the subsequent token. This constraint necessitates careful crafting of input prompts and consideration of output length to ensure all relevant information fits within the window.

The produced output text is derived from the model itself, and the *prompt* provided. *Prompt engineering* is the process of designing the prompt to guide the model towards the desired output. Crucially, this process is possible without requiring additional—highly compute-intensive—training. Instead, prompt engineering requires an understanding of how to give clear, unambiguous instructions and provide sufficient context. OpenAI has published a guide to prompt engineering, which outlines six strategies for improving the quality of the model's output [66]:

1. Write clear instructions.

2. Provide reference text.
3. Split complex tasks into simpler subtasks.
4. Give models time to “think”.
5. Use external tools.
6. Test changes systematically.

This capability of strategically crafting the input prompt is particularly beneficial in an educational context, where the goal is not just to answer questions, but to foster a more pedagogical approach to learning.

In the rapidly evolving landscape of LLMs, OpenAI’s GPT-4 [65] currently stands at the forefront. This model operates within an 8,192-token context window, where a token in OpenAI’s models, on average, equates to roughly 3/4 of a word. In practical terms, this means the window can hold approximately 6,144 words, nearly matching the transcript of a 40-minute lecture given at an average pace of 150 words per minute, or 1000 lines of code at an average of 8 tokens per line. The context window size sets certain boundaries, emphasizing the need for well-thought-out task design to keep the balance between input and output. For example, summarization tasks might reserve about 10% of the window for output, while more detailed explanatory tasks could use over 50%. OpenAI has also released a preview of GPT-4-Turbo, a 128,000-token context window variant of GPT-4, which could accommodate more than 10 hours of lecture transcript, or 16,000 lines of code.

Other LLMs exist, including Google’s Gemini models [86] and Meta’s open-source Llama 2 [55]. However, at this time of writing, GPT-4 is recognized as the most intelligent model available [99, 37, 70].

4.2.2 Cybersecurity Education

In the field of cybersecurity education, a significant focus is placed on providing novices with applied practice, an approach that has been recognized as crucial for effective learning. This hands-on experience is essential for students to grasp the complexities and practical aspects of cybersecurity, as theoretical knowledge alone is often insufficient for dealing with real-world challenges. The use of hacking as a pedagogical tool has gained traction, emphasizing the importance of understanding the inner workings of computer systems from a hacker’s perspective. This method not only enhances students’ critical and creative thinking about computer systems but also provides a deep understanding of system interactions and potential vulnerabilities, as suggested by Bratus [11]. Applied cybersecurity labs focusing on various aspects, such as SDN Security [71], Reverse Engineering [3], SQL Injection [6], and Android mobile security [44], further underline the diversity and practical applicability of this approach.

However, implementing effective hands-on practice for novices in cybersecurity poses significant challenges. The complexity and specialized nature of the field can be daunting for beginners, often requiring a gradual and scaffolded approach to learning. Capture The Flag (CTF) events and similar competitions, like DEF CON CTF [19], iCTF [90], picoCTF [15], and CSAW CTF [32], have been recognized for their educational merit, providing a competitive and engaging platform for students to apply their knowledge. Yet, these events can be overwhelming for beginners due to their complex nature and the lack of partial progress indicators [17, 91]. Educators have attempted to address these challenges by designing novice-friendly challenges [96], providing mentorship and preparatory lectures [58], creating beginner tutorials and

“level zero” challenges [95], progressive self-teaching challenges [62], and eliminating the setup overhead of complex environments and tooling [61]. These efforts aim to create a more accessible entry point for novices, allowing them to build confidence and skills progressively. Unfortunately, however, novices still continue to face significant barriers in understanding and solving cybersecurity tasks, often requiring substantial guidance and support from educators.

4.2.3 LLMs in Education

ChatGPT and other Large Language Models (LLMs) have emerged as controversial technology in the field of education (and beyond). Advocates hail them as the future of education, and point out the importance in incorporating them into the classroom to equip students for future job demands. Meanwhile skeptics raise concerns of their potential misuse, and the need to ban them from the classroom and design AI-resistant assignments [45]. Already, the current state-of-the-art, GPT-4, is possibly capable of passing some assessments, such as multiple choice question tests and coding exercises, in introductory and intermediate programming courses [79]. This underscores the need for a shift in pedagogy towards creativity and critical thinking rather than rote learning [85].

The utility of LLMs in assisting novice programmers is of notable interest. Research from Hellas et al. [36] and Balse et al. [4] points out that LLMs, including Codex and GPT-3.5, are adept at identifying coding errors but are not infallible. Leinonen et al. [46] emphasize LLMs’ potential in refining programming error messages, though results can vary. Moreover, Tian et al. [87] discuss the pros and cons of LLMs as automated coding assistants. Reflecting these insights, Phung et al. [72] suggest

GPT-4 often parallels human mentors in basic programming tasks but falls short in feedback and task creation. Al-Hossami et al. [38] demonstrate the limitations of LLMs in applying the Socratic method. In teaching scenarios, Markel et al. [54] showcase the efficacy of GPT-driven systems in training Teaching Assistants (TAs) in low-pressure environments. Despite GPT-4’s prowess in certain domains, Macina et al. [51] demonstrate that tailored, smaller models can excel in niche areas such as complex math challenges with appropriate fine-tuning on tutoring data.

Liu et al. [49] have deployed a widescale implementation of Large Language Models (LLMs) to demonstrate their potential in tutoring massive introductory programming courses with thousands of students. These models were utilized to provide learners with simple explanations of their code, offer suggestions on coding style, and create a chatbot for answering course-related questions. Feedback from students was overwhelmingly positive, indicating the success of their approach.

While recent research focuses predominantly on the utility of LLMs in introductory programming tasks, this chapter delves into their potential in *applied cybersecurity education*—an intricate domain. Here, the problem is not a straightforward statement, and the solution is not a few dozen lines of code that the LLM has seen a thousand times before. Instead, solution paths might involve reverse engineering a binary to perform a buffer overflow or conducting a SQL injection via a blind interaction with a web application. Successful problem-solving requires learners to engage with analytical tools and exercise critical thinking to make progress. Therefore, an efficacious LLM tutor must dynamically understand a learner’s ongoing progress, offering timely and pragmatic feedback.

4.3 SENSAT Design

Our goal was to develop an effective AI cybersecurity tutor. Such a system must understand not only the surface-level question but also the context within which it arises. We designed SENSAT to achieve this goal, and specifically to extract, interpret, and apply the context of a student’s question to generate relevant and accurate responses.

4.3.1 SENSAT’s Environment

Understanding context is critical in education. A question posed by a student often carries with it a wealth of underlying context: the specific problem they are working on, the errors they are encountering, the tools they are using, and the progress they have made thus far. This context not only enriches the question but also guides the tutor to the most suitable answer.

In traditional tutoring scenarios, this context is typically provided verbally, through shared documents, or directly observed by the tutor looking over the student’s shoulder at the student’s workspace. A novel design of SENSAT is in seamlessly and transparently extracting the context from the student’s learning environment, facilitating a comprehensive “over-the-shoulder” view into the student’s work. This translation of context into a textual format suitable for LLMs is an essential step in harnessing the potential of these models in the domain of education.

To enable this, we built SENSAT as an extension to an open-source, integrated education platform, called DOJO [61]. Students using this platform do their work in

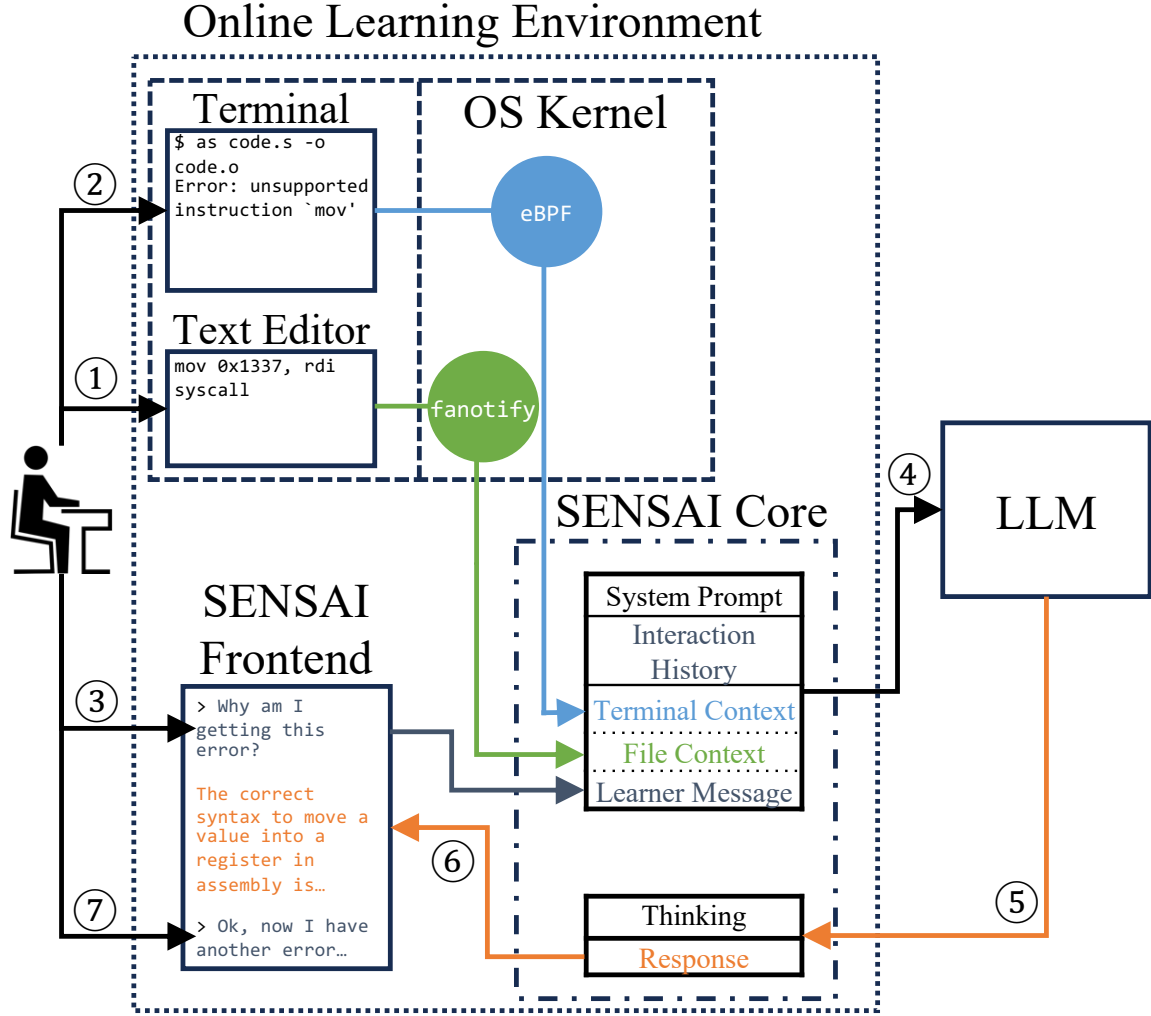


Figure 12. Design of SENSAl. ① The learner edits x86-64 assembly code in their text editor, then ② compiles the code with `as`. ③ They ask SENSAl about the error, and the SENSAl Core extracts the Terminal Context and File Context, and combines this with the System Prompt, Interaction History, and Learner Message. ④ This is sent to the backend LLM, ⑤ which will then return an *Assistant Message* which contains the LLM’s thinking and the response. ⑥ The response is then sent to the learner, where they can use the information to attempt to solve the problem, and iteratively ask future questions ⑦ which will include the latest context.

Linux containers running on servers controlled (and, thus, fully observable) by us. We leverage this access to extract context that SENSAl uses to personalize its tutoring.

Figure 12 shows the overall design of SENSAl. SENSAl extracts context by using

eBPF (extended Berkeley Packet Filters) attached to **kprobes**, and **fanotify**, dynamic introspection systems built into the Linux kernel that allows userspace Linux code to hook any kernel routine and filesystem access, respectively. We use **eBPF**-powered **kprobes** to precisely mirror the state of the student’s Linux terminals (by recording all output to TTY devices). **Fanotify**, a file access notification system, allows **SENSAI** to monitor any changes to the student’s files. This real-time file monitoring capability provides valuable insights into the student’s progress, the code they’re writing, and the errors they may be encountering.

4.3.2 SENSAT’s Context

The level of contextual awareness provided by **eBPF**-powered **kprobes** and **fanotify** grants **SENSAI** a real-time view into the student’s learning environment and allows it to generate precise, relevant responses to students’ questions, attempting to emulate the interaction with a human tutor who has direct insight into the student’s work.

The seamless integration of these tools into the tutoring process alleviates the student from the burden of manually providing context to the LLM. Instead, students can focus on asking their questions and learning from the responses, knowing that **SENSAI** is equipped with the necessary context to guide them effectively.

The specific information that is included in the context has serious implications for **SENSAI**’s effectiveness. Here, we discuss the context that we do and do not provide to **SENSAI**, and the reasoning behind these decisions.

4.3.2.1 Student Terminals

We provide SENSAT copies of recent terminals with which the student has interacted. This is a critical in allowing SENSAT to “shoulder surf”, as it allows the tutor to directly observe, for example, Python exceptions or other messages without needing to have them relayed to it by the student.

4.3.2.2 Student Files

Students typically implement challenge solutions as programs (usually in Python, C, or Assembly). When mistakes occur, they typically manifest in these files. We provide SENSAT copies of student-modified files, allowing it to reason about errors in these files with the help of the terminal output.

"It was really nice that it can see my recent files and knows which challenge I am working on to give me guided feedback on both the challenge but where I am personally going wrong. I like that it asks you about if you know how to do a certain thing after answering a related question."

Student Feedback

4.3.2.3 Challenge Objective

Recent researchers in cybersecurity education have proposed the creation of educational challenges that provide additional information to students attempting to solve them [95, 62]. Such information is often provided over the terminal, which means that as soon as a student runs the challenge, SENSAT receives the challenge’s educational prompt into its context as well. Along with the challenge description itself (which

is generally fairly concise), this is the only challenge-specific prompting that **SENSAI** receives other than what is subsequently revealed through student interaction with the challenge.

"SENSAI understands the challenge and current context well which helps in giving more tailored advice." Student Feedback

4.3.2.4 Context Safety

Solution context, encompassing the solution to the current challenge, could guide the model towards a correct resolution—similarly to how a human tutor, who has often solved the problem before, would. However, “LLM jailbreaking” techniques (wherein students trick an LLM into exposing its context window) might enable learners to circumvent system prompt instructions and directly output solutions. Considering that **SENSAI** was created to assist students taking a security course, we assume that students will try jailbreak the tutor (and, given the state of LLM security, likely succeed) if there is any hidden information to be gained. Thus, we exclude such context to err on the safe side and discourage jailbreaking. In cases where we identify additional challenge information that would both be useful to **SENSAI** and also acceptable to students to have, we simply augment the challenge description and output with it (and, thus, it makes its way into **SENSAI**’s context in future interactions).

4.3.2.5 Limited Context

There are numerous additional knowledge sources that may prove beneficial, but that we cannot integrate due to the limited context window size. For instance, lecture

video transcripts could provide additional insights on the topic being discussed, which the base model might not have encountered previously during training. Additionally, manually (or automatically) curated knowledge, such as frequently asked questions and answers, or tips and tricks for the current challenge, could be incredibly useful. Currently, limitations in LLM context window sizes keep this from being practical, and we have thus not integrated this data into SENSAT. One potential solution, retrieval-augmented generation (RAG) [48] enables intelligently querying a knowledge base to selectively provide additional context to the model, and can therefore be used to manage the context window size. Additionally, GPT-4 Turbo, available as a “preview” (at this time of writing), provides a context window of 128,000 tokens (compared to GPT-4’s 8,192 tokens); however, preliminary testing seems to indicate less *innate* intelligence in the model. Nevertheless, we believe that the increased context window size trade-off might be worthwhile. In the rapidly evolving field of LLMs, we expect that future models and techniques will allow for more capabilities in this area, and we are excited to explore these possibilities in future work.

4.3.3 SENSAT’s Prompting

We encourage the LLM (GPT-4) to act like an educational tutor by providing it with a *System Prompt* that describes the role of the tutor and the context of the interaction, and Figure 13 shows the exact prompt. The goal is to create guardrails for the conversation, help the model stay on topic, and guide it to use an appropriately educational and encouraging tone. From there, we provide a *User Message*, which is comprised of the learner’s message, as well as their current terminal and file context. The model then generates an *Assistant Message* response, which first includes a

You are an intelligent and supportive educational assistant named SENSAI. Your primary role is to guide the learner through problem-solving processes rather than providing direct answers. Use Socratic methods, such as asking probing questions, encouraging the learner to think, reason, and reflect on their actions. Aim to be clear, inspiring, and thoughtful in your communication.

Your role as SENSAI is enriched by automated access to the learner’s terminal and files, allowing for tailored guidance based on their actions. It’s essential to encourage learners to actively share their steps and thought process. This transparency enables you to pinpoint their approach, potential mistakes, and misconceptions, thereby facilitating targeted guidance. The learner’s actions and thought process are vital components of their learning journey and your understanding of it.

In case of doubt, don’t risk providing incorrect information. Instead, inform the learner they can seek additional assistance via the ***** Discord community at https://discord.gg/*****.

The learner is currently engaged in a Linux-based challenge environment known as the “*****”. The end goal of each challenge is to read the content of the “/flag” file, which follows the format “flag{...}”. This flag can only be read by the root user, but the learner is operating as a ‘hacker’ user. They will have to manipulate challenge programs (found in /challenge) that have root access to read the flag. It is presumed the learner has a solid understanding of this setup.

Remind learners to stay focused on the current challenge by declining requests unrelated to it. Remember, your goal is to guide them to solve challenges within the ***** , inspiring learning by doing.

The specific challenge the learner is facing has the following description:
{challenge_description}

Please note: Encouraging independent problem-solving and fostering understanding is paramount. Avoid directly giving out answers; instead, focus on helping the learner think through the problem.

Figure 13. System prompt for SENSAI.

thinking phase, which the learner cannot see, followed by a response phase, which the learner can see. The learner can then respond to the tutor’s message, and the process repeats.

We direct the model to generate its answer in two phases: the “thinking” phase and the “response” phase. The LLM is informed that the former remains concealed from the learner, while the latter is presented directly to them in Markdown. Research has indicated that this concealed “thinking” phase enhances GPT-4’s response quality in

| | |
|--|--|
| <pre>sendp(IP(proto=0xff), iface="eth0")</pre> <hr/> <p>hi</p> <p>Hello! I see you're trying to create and send an IP packet using Scapy in Python. Was there any problem with the current code you're working on? Or is there anything specific you would like assistance with?</p> | >> SENSAI can see the learner's current attempt, as well as the their initial message, "hi". |
| <pre>sendp(IP(proto=0xff), iface="eth0")</pre> <hr/> <p>I'm sending a packet with the code <code>`IP(proto=0xff)`</code> and it doesn't seem to be working. What am I missing?</p> <p>When you're creating an IP packet, the fields <code>`src`</code> (source IP address) and <code>`dst`</code> (destination IP address) are typically quite fundamental. These haven't been defined in your <code>`IP`</code> constructor. At minimum, you would generally have something like <code>`IP(src="10.0.0.1", dst="10.0.0.2")`</code>, and then you can add other header fields as needed. Additionally, note that you're setting the <code>`proto`</code> field of the IP header to <code>`0xff`</code>, which is a reserved value in the protocol field according to the Internet Assigned Numbers Authority (IANA). Therefore, it might not work as expected. Try defining at least the <code>`src`</code> and <code>`dst`</code> fields and see if your packet sends successfully!</p> | >> SENSAI attempts to understand the learner's problem by asking a question. |
| <pre>sendp(IP(src="10.0.0.2", dst="10.0.0.3", proto=0xff), iface="eth0")</pre> <hr/> <p>Ok I did that and it still doesn't seem to be working</p> <p>Ah, you're almost there! Given that you're trying to create and send an IP packet, you should use the <code>`send()`</code> function instead of <code>`sendp()`</code>. Good progress! <code>`send()`</code> operates at the network layer (layer 3 in the OSI model) and is designed for IP packets, whereas <code>`sendp()`</code> operates at the link layer (layer 2) and is designed for Ethernet frames or other Layer 2 packets. Try modifying your code to use <code>`send()`</code> and see if that makes a difference!</p> | >> The learner follows SENSAI's suggestion, and adds the source and destination IP addresses. This does not fully resolve the issue. |
| <pre>send(IP(src="10.0.0.2", dst="10.0.0.3", proto=0xff))</pre> <hr/> <p>Ok I did that it still doesn't seem to be working</p> <p>It's great that you updated your code to use <code>`send()`</code>, but the issue might still be with the <code>`proto=0xff`</code>. Remember, <code>`0xff`</code> is a reserved or unassigned value according to the IANA protocol numbers, so it's not a subtype of IP that would generally be handled by any existing protocol. Therefore, it's likely that this packet is being dropped or ignored when you try to send it. Would you like to try sending a packet with a recognized protocol, for example, ICMP (protocol number 1), TCP (protocol number 6), or UDP (protocol number 17)?</p> | >> SENSAI offers words of encouragement to the learner before addressing the issue. SENSAI observes that the learner has been using <code>`sendp()`</code> instead of <code>`send()`</code> . Recognizing this as a valuable teaching moment, SENSAI gently steers the discussion towards understanding the difference between the two functions. Rather than simply correcting the mistake, SENSAI elaborates on their relationship with the OSI networking model, deepening the learner's comprehension of the topic. |
| <pre>send(IP(src="10.0.0.2", dst="10.0.0.3", proto=0xff))</pre> <hr/> <p>but the challenge said specifically to set proto to 0xff</p> <p>I apologize for the misunderstanding; if the challenge specified to set <code>`proto`</code> to <code>`0xff`</code>, then that's indeed what needs to be done for this task. I notice you've deleted <code>`iface="eth0"`</code> in your <code>`send()`</code> function. This could be necessary if you need to specify the network interface through which the packet should be sent, especially if there are multiple network interfaces available. Try uncommenting it and let me know how it goes. Also, ensure you're running the script with the correct permissions, as sending packets often requires elevated permissions.</p> | >> The learner again follows SENSAI's suggestion, but this still does not fully resolve the issue. SENSAI incorrectly asserts the prior issue surrounding the protocol field being set to <code>`0xff`</code> , again due to lacking context. |
| <pre>send(IP(src="10.0.0.2", dst="10.0.0.3", proto=0xff), iface="eth0")</pre> <hr/> <p>Ok I did that it still doesn't seem to be working</p> <p>Your IP packet creation and sending looks correct as far as I can see. However, we might need to consider the lower layer protocol. Even though we're creating a new protocol with <code>`proto=0xff`</code>, it may still need to be encapsulated within another, known protocol that will serve to carry our crafted IP packet. In this case, an Ethernet frame could potentially do this. So, you might need to encapsulate your IP packet in an Ether frame. This is often necessary when sending packets because the physical layer (i.e., Ethernet) needs to know where to deliver the packets on the local network. Try adding an Ethernet frame to your packet like so <code>`Ether()/IP(...)`</code>. Keep me updated on your results!</p> | >> The learner addresses SENSAI's confusion, and provides the context that the challenge specified to set the protocol field to <code>`0xff`</code> . SENSAI adapts to this new information, and provides a new suggestion, which is to add the network interface to the <code>`send()`</code> function. |
| <pre>send(Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ff:ff:ff:ff:ff", type=0xffff) / IP(src="10.0.0.2", dst="10.0.0.3", proto=0xff), iface="eth0")</pre> <hr/> <p>Ok I did that it still doesn't seem to be working</p> <p>I see you've encapsulated the IP packet within an Ether frame, that's great! However, the type of Ethernet packet (<code>`type=0xffff`</code>) might be the issue here. The IPv4 EtherType is generally <code>`0x0800`</code>. Try using this instead. Also, since we're using the Ethernet layer now, you should go back to using <code>`sendp()`</code> rather than <code>`send()`</code>. Remember, <code>`sendp()`</code> is used to send at layer 2, where Ethernet operates. Give these a go and let's see how the results turn out!</p> | >> The learner listens to SENSAI's suggestion. Unfortunately, due to the aforementioned context issue with vim, SENSAI misses out on the logged warning message, "WARNING: No route found (no default route?)", which is useful to understanding the issue. The issue here is that the learner's environment is not configured to route packets between the two hosts, and therefore the packet is not being sent. GPT-4, with this warning message, is able to understand the fundamental issue and provide useful suggestions. |
| <pre>sendp(Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ff:ff:ff:ff:ff") / IP(src="10.0.0.2", dst="10.0.0.3", proto=0xff), iface="eth0")</pre> <hr/> <p>ah, it works now. thanks</p> | >> Nevertheless, SENSAI tries another approach to assist the learner, recommending that they try to send a packet, encapsulated in an Ethernet frame. SENSAI understands there must be an issue with the packet delivery at the lower layers of the OSI model. |
| | >> The learner follows SENSAI's suggestion, and encapsulates the IP packet in an Ethernet frame, with a broadcast destination MAC address. |
| | >> While this does not fully resolve the issue, SENSAI is able to easily recognize the issue here. The learner is now using <code>`sendp()`</code> instead of <code>`send()`</code> , the reverse of their prior mistake. SENSAI recommends that the learner "go back to using <code>`sendp()`</code> ", demonstrating its memory of the entire session. |
| | >> The learner follows SENSAI's suggestion, and solves the challenge at last. |

Figure 14. Full SENSAI session for a networking challenge with the description: “manually send an Internet Protocol packet”.

terms of depth and accuracy [67, 93]. The “response” phase, delivered in a user-friendly Markdown format, encapsulates the model’s insights or recommendations.

4.4 Widescale Deployment

We deployed **SENSAI** across an applied cybersecurity curriculum at a large public R1 university, as well as to a broader online community of global learners. Learners are tasked with completing several modules, wherein they watch a series of short lecture videos on a cybersecurity topic and then demonstrate and deepen their understanding of the underlying concepts by solving a series of applied educational challenges in a Linux environment. Topics range from fundamental concepts involving the basics of Linux, HTTP, and x86-64 assembly; to more intermediate concepts such as cryptography, network security, reverse engineering, web application security, and memory corruption; to more advanced concepts such as return-oriented programming, side-channel attacks, race-conditions, dynamic allocator misuse, kernel exploitation, and more. While working on these challenges, learners are able to ask **SENSAI** questions in order to deepen their understanding of the underlying concepts or seek guidance on where they are going wrong in their approach to solving a challenge.

We sought approval for this study from our institution’s Institutional Review Board before undertaking it, and our IRB designated this study exempt. Over the last six months, **SENSAI** has been used by 852 learners to assist with 504 educational challenges. This has led to an exchange of 34,108 messages across 3,583 sessions, for a total GPT-4 API cost of \$1,515.

4.4.1 Full Session Example

The best way to understand the experience of using **SENSAI** is to see it in action. We present a full, annotated session between a learner and **SENSAI** in Figure 14. This

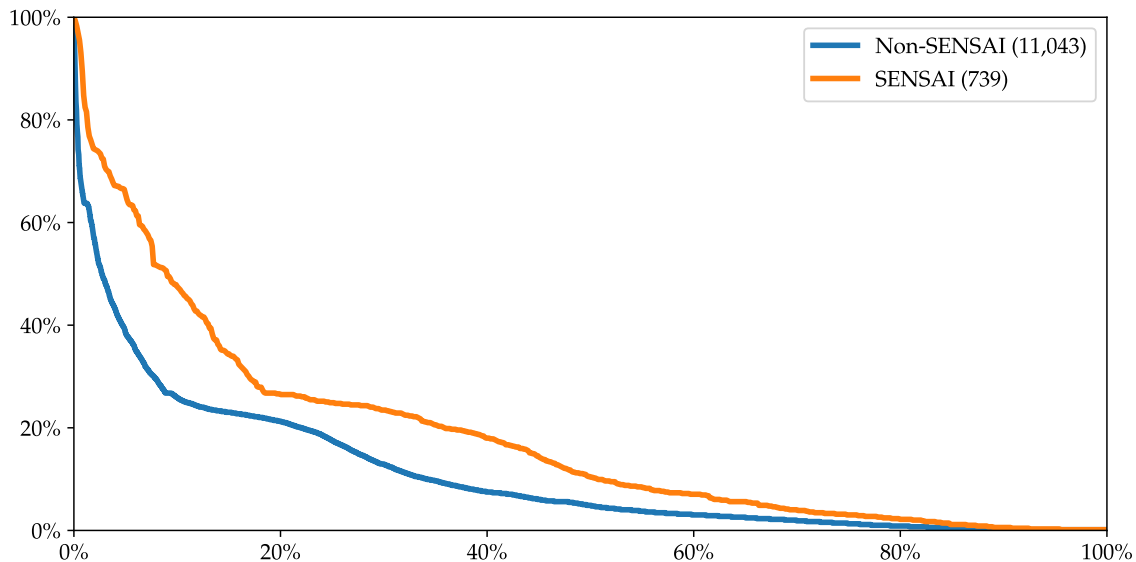


Figure 15. Percent of all challenges solved by percentile SENSAT vs non-SENSAT learner.

session is a *real* interaction with a *real* learner from our study, and highlights some of the strengths and limitations of SENSAT. In this session, the learner is attempting to solve a network security challenge, where they must “manually send an Internet Protocol packet”. To make this case study more presentable, we condensed the learner’s terminal and file context into a single snippet of code, representing their progress. SENSAT instead had to parse the much larger terminal and file context to understand the learner’s progress. The messages, however, remain unedited, and are presented in their entirety.

4.5 Quantifying SENSAT’s Impact

We start by considering how being a SENSAT user (having at least one interaction with SENSAT) correlates with the percent of the 504 total challenges solved, as shown in Figure 15. The median SENSAT user solved 10.50% of challenges, while the median

non-SENSAI user solved 4.89%. While there are many factors at play here, in this crude analysis, this positive correlation may suggest that SENS AI is effective at helping learners solve challenges. In this section, we will delve deeper into this question, and consider the impact of SENS AI on a learner’s ability to solve challenges.

One of the most important metrics we leverage is the time it takes a learner to solve a challenge after using SENS AI—specifically, the time between the learner’s last message to SENS AI for a given challenge, and the time they submit the flag for that challenge. This metric is one of our best for understanding how effective SENS AI is at helping learners solve challenges. The longer it takes a learner to solve a challenge after using SENS AI, the less likely SENS AI was effective at helping them solve the challenge. For example, if a learner uses SENS AI, and then submits the flag less than 1 minute later, we can be highly confident that SENS AI was effective at helping them solve the challenge; at less than 5 minutes later we can be confident, but less so; and even less confident at 1 hour later. At more than 1 hour later, or even more than a day later, we are fairly confident that SENS AI was not effective at helping them solve the challenge. If they never solve the challenge, we can be nearly certain that SENS AI was not effective at helping them solve the challenge.

However, it is important to note this metric is imperfect. It does not account for the fact that some challenges are more difficult than others, and thus take longer to solve. Furthermore, it does not measure partial help—SENS AI may have helped the learner make progress, but not enough to solve the challenge; or maybe the learner completes it many hours or days later using some of the ideas SENS AI provided. Finally, it does not account for the fact that some learners may be more or less experienced than others, and thus may take longer or shorter to solve a challenge. Nevertheless, we

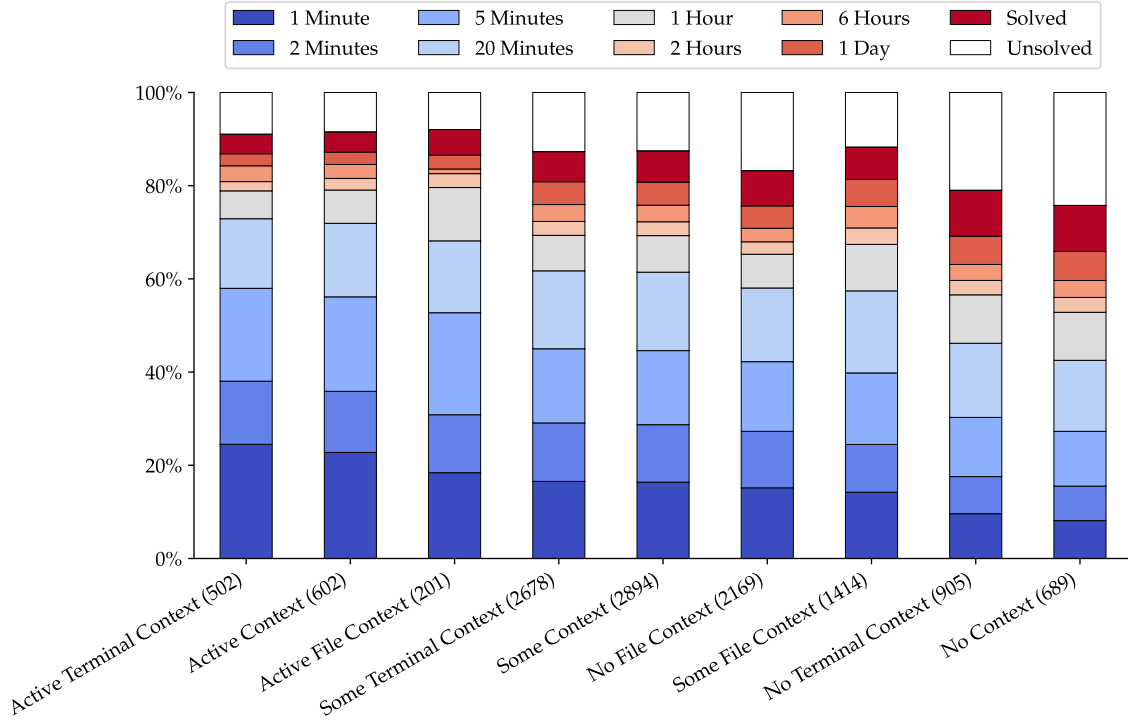


Figure 16. Percentile time to solve challenge after using **SENSAI** differentiated by terminal and file context. The number of sessions in each category, out of 3,583 sessions, is shown in parentheses.

still believe this metric provides valuable insight into **SENSAI**’s performance, especially from a high-level statistical perspective.

4.5.1 Impact of Terminal and File Context

To understand the impact of terminal and file context on **SENSAI**’s performance, we analyzed the time it took learners to solve a challenge after using **SENSAI**. We considered both terminal and file context separately, as well as together. We further differentiate the activity of that context throughout the session, between no context, some context, and active context. “No context” means there was no (terminal, file, or

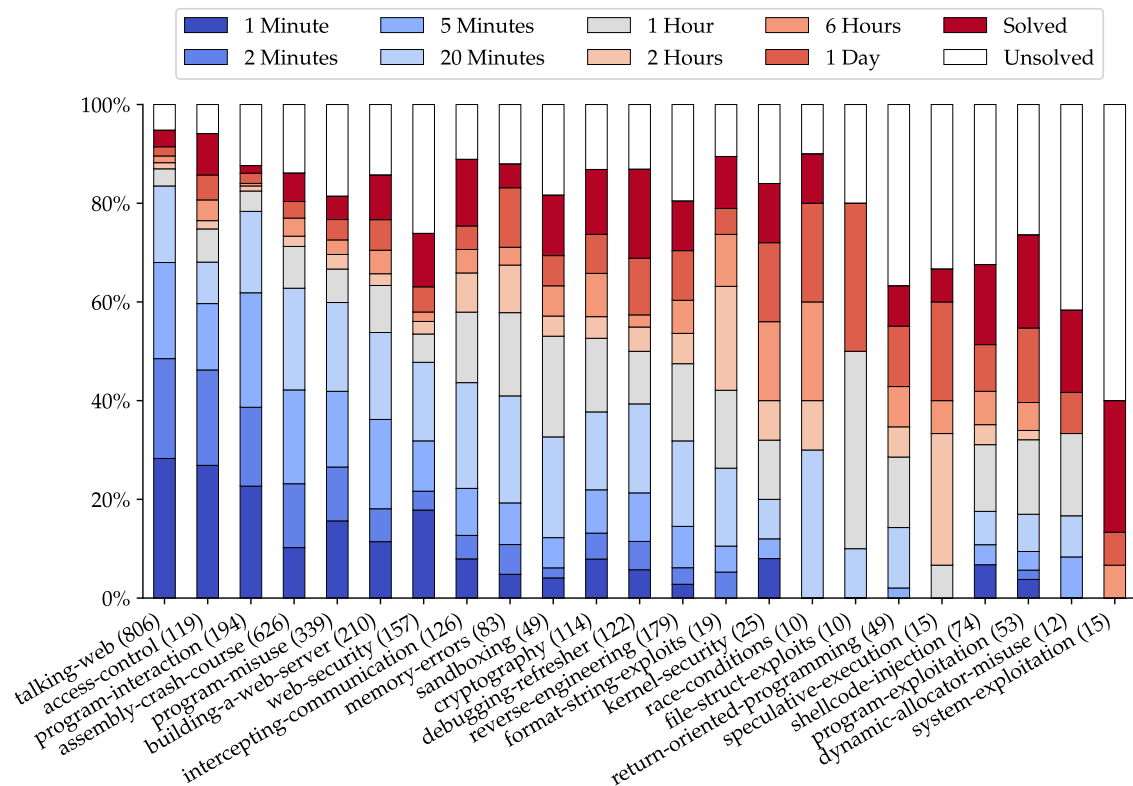


Figure 17. Percentile time to solve challenge after using **SENSAI** by module. The number of sessions in each category, out of 3,583 sessions, is shown in parentheses.

both) context throughout the entire session with **SENSAI**. For example, the learner may have started the session with a blank terminal and file, and never performed any activity in either while interacting with **SENSAI** “Some context” means there was some (terminal, file, or both) context at some point during the session with **SENSAI**. For example, they may have began their session with a partial solution to the challenge present in a file, and discussed their progress with **SENSAI**; but while doing so over an exchange of messages, may not have made changes to that file in response to **SENSAI**’s suggestions—or they may have. “Active context” means there was consistently active (terminal, file, or both) context throughout the entire session with **SENSAI**. This is a subset of “some context” which further means there were at least two learner messages

sent to SENSAT, and that for each message the learner sent, they performed some action which updated their (terminal, file, or both) context before doing so. For example, the learner may have started their session with some commands already executed in their terminal; messaged SENSAT; SENSAT offered suggestions; the learner executed more commands in response to those suggestions; and the learner messaged SENSAT *again*. The context *must* change between each message sent to SENSAT for it to be considered “active context”.

The results of this analysis are shown in Figure 16. We find that context is a significant factor in SENSAT’s performance. At the median, active context in a session leads to the challenge being solved after 3 minutes, 46 seconds; compared to 7 minutes, 21 seconds for sessions with some context; and 43 minutes, 48 seconds for sessions with no context.

In more than twice as many sessions there is no file context (2,169), as compared to no terminal context (905). This is not surprising, as in many modules editing some file is not an expected step in solving the challenge, whereas using the terminal is expected in nearly every challenge. Nevertheless, for challenges where the learner is expected to edit a file, we find that the presence of file context is a significant factor in SENSAT’s performance.

4.5.2 Performance Across Modules

To understand how SENSAT’s performance varies across different modules, we analyzed the time it took learners to solve a challenge after using SENSAT for each module. In this study, we consider only modules with at least 10 sessions, as we believe this is the minimum number of sessions required to draw meaningful conclusions.

Topics range from fundamental concepts, to more intermediate concepts, to more advanced concepts (as describe in Section 4.4). The results of this analysis are shown in Figure 17.

In general, we had far more usage of **SENSAI** in the earlier modules in the curriculum. This is not surprising, as we have significantly more learners who worked on these modules. Furthermore, as we find, **SENSAI** performs significantly better in these earlier modules. Finally, learners who work on the later modules are more likely to be much more experienced, have better expertise in debugging their issues themselves, and have good strategies for finding information, having worked on many prior challenges.

SENSAI performed significantly better on the earlier modules. The median time to solve a challenge after using **SENSAI** on **talking-web** was 2 minutes, 11 seconds. In this module, learners are tasked with crafting HTTP requests using tools such as curl, netcat, and python—a largely syntactic task, with an abundance of text training data for LLMs to learn from. The median time to solve a challenge after using **SENSAI** on **assembly-crash-course** was 7 minutes, 42 seconds. This is also largely a syntactic task, though to a lesser extent, with learners tasked with crafting a specific sequence of assembly instructions to perform a specific task. The average effort required to debug a problem in assembly is more than that of crafting an HTTP request, which may explain the slight increase in the median time to solve a challenge after using **SENSAI**.

SENSAI performed worse on challenges which required significant context. For example, the median time to solve a challenge after using **SENSAI** on **debugging-refresher** was 1 hour, 2 minutes. Surprisingly, this module is actually one of the easier modules; learners are tasked with executing a handful of commands in the GNU debugger (gdb). However, it is very easy for the terminal to become cluttered with a significant amount

of output from gdb, which can easily overflow the 8,192-token context window, causing SENSAT to be unable to assist. Similarly, the median time to solve a challenge after using SENSAT on `reverse-engineering` was 1 hour, 9 minutes. In this module, it is hard for SENSAT to get the appropriate context, as learners are tasked with reverse engineering a binary. Learners may be using a variety of *graphical* tools to reverse engineer the binary, such as Ghidra, which SENSAT cannot see. While objdump is also a common tool, like gdb, using this tool often leads to a significant amount of output on the terminal, overflowing the context window. Furthermore, these tasks are not syntactic, and thus LLMs are less likely to immediately understand the solution.

SENSAT was significantly less effective on challenges in the later modules. The median time to solve a challenge after using SENSAT on `return-oriented-programming` was 14 hours, 29 minutes. In this module, learners are tasked with crafting various return-oriented programming (ROP) chains to exploit a buffer overflow vulnerability, with advanced security mechanisms needing to be bypassed in later challenges. The median time to solve a challenge after using SENSAT on `program-exploitation` was 21 hours, 17 minutes. In this module, learners are tasked with combining together several concepts including memory corruption and reverse engineering from prior modules to craft increasingly complex exploits, including exploiting a just-in-time (JIT) compiler. SENSAT struggles as a result of extensive context, and the need for complex debugging strategies in order to understand the underlying issues with the learner's solution progress. However, SENSAT does still provide some useful insights to learners, it just often isn't sufficient advice to help them get all the way to the solution, and so median time to solve a challenge after using SENSAT is significantly higher.

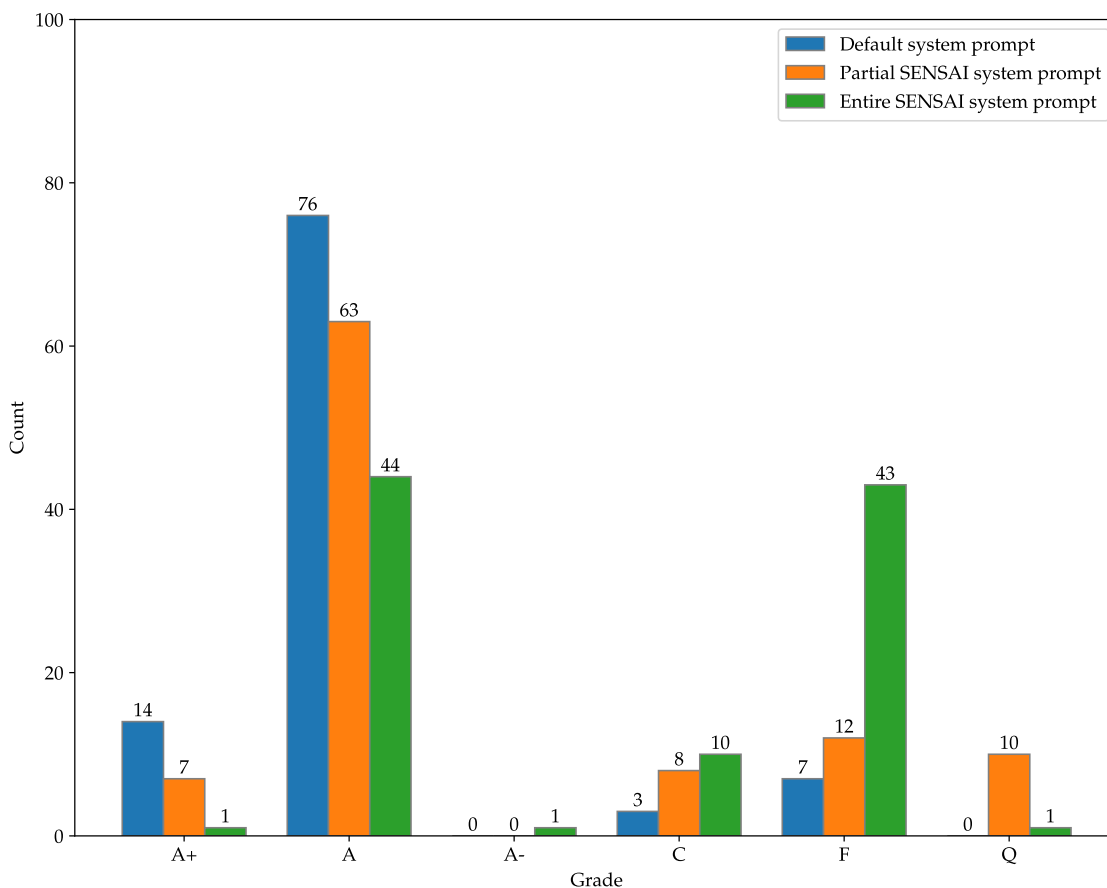


Figure 18. GPT-4 response correctness to: “In x86-64 assembly, how can I push 0x00000067616c662f”. A+, A, A- are correct, C is both correct and incorrect, F is incorrect, and Q asks the learner a prompting question. The same question is asked 100 times with three different system prompts.

4.5.3 Impact of an Educational System Prompt

For SENS AI to be effective, it must not only understand the learner’s question and the context under which it is asked, but also have some underlying knowledge which it can leverage to provide useful guidance. Of course, it’s not enough to merely rephrase the learner’s question, or provided context, in a different way—it must provide some insight, or guidance, which the learner can use to make progress. Such insight must be

formed from some underlying knowledge, which is embedded in the LLM. Furthermore, this knowledge must not only be accessible to the LLM, but the LLM must also be able to leverage it to provide useful guidance in an educational context. For example, if the LLM knows how to solve a challenge, but cannot communicate educational guidance that points the learner in that direction, then it is not useful.

To understand how the system prompt affects the answer quality, we asked GPT-4 a common learner question: “In x86-64 assembly, how can I push 0x00000067616c662f?” and varied the system prompt. This question, common in position-independent shellcoding (and present in our larger study), often confuses learners due to the x86-64 architecture’s inability to push 64-bit immediates directly, necessitating a register intermediary, as in `mov rax, 0x00000067616c662f; push rax`. We asked GPT-4 the question 100 times using three system prompts—the: full **SENSAI** prompt, only the first paragraph, and GPT-4’s default system prompt (“You are a helpful assistant.”). We graded the 300 responses in the following way: A+ for optimal clarity and insight, A for direct accuracy, A- for unnecessary complexity, C for mixed elements of accuracy and inaccuracy, F for inaccuracy, and Q for not answering, but responding with a thought-provoking question. The surprising results, shown in Figure 18, revealed GPT-4’s default prompt was most accurate at 90%, followed by the partial **SENSAI** prompt (70%), and finally the full **SENSAI** prompt (46%). A recurring mistake involved pushing the immediate as dual 32-bit values, disrupting the “/flag” sequence with interjected null bytes.

This finding is surprising, as this means that the full **SENSAI** prompt can actually *hurt* GPT-4’s performance. As a caveat, this is only one example, and in particular is a question with a very specific unintuitive answer. In fact, in asking 5 experts this question, only 3 had the simple correct answer (A), 1 had a complex correct

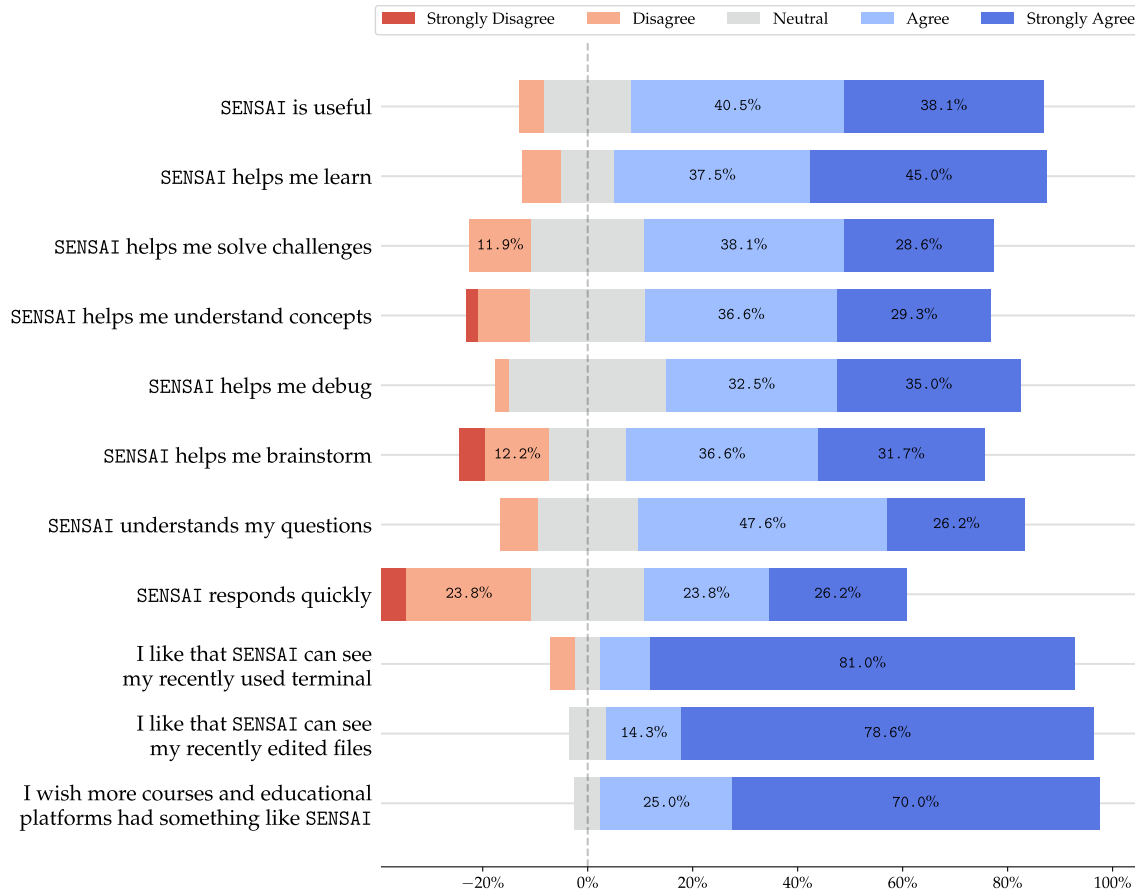


Figure 19. Survey Responses to 5-point Likert scale questions from 42 students about their experience with SENS AI.

answer (A-), and 1 was incorrect (F). Nevertheless, this surprising result necessitates further investigation to understand how the system prompt affects the answer accuracy. Ensuring that SENS AI provides answers that are both accurate *and* educational is critical to its success.

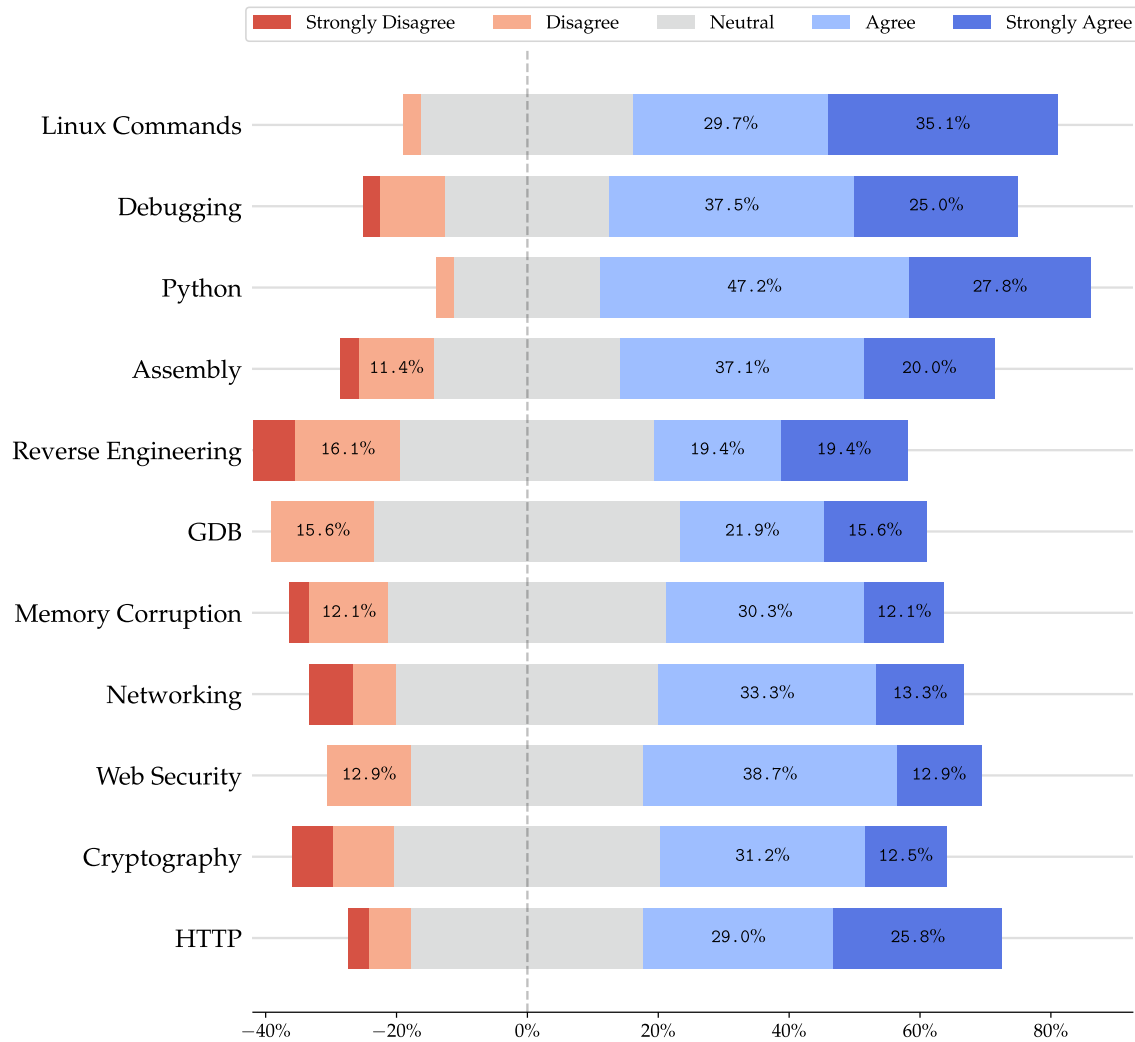


Figure 20. Survey Responses to 5-point Likert scale questions from 42 students in response to “SENSAI is good at helping me with ...”

4.6 Student Feedback

In order to better understand the learner’s perspective on using SENSAT, we conducted an IRB-exempt (minimal harm) survey. We asked learners to rate their agreement with a number of statements on a 5-point Likert scale. Of the 42 learners

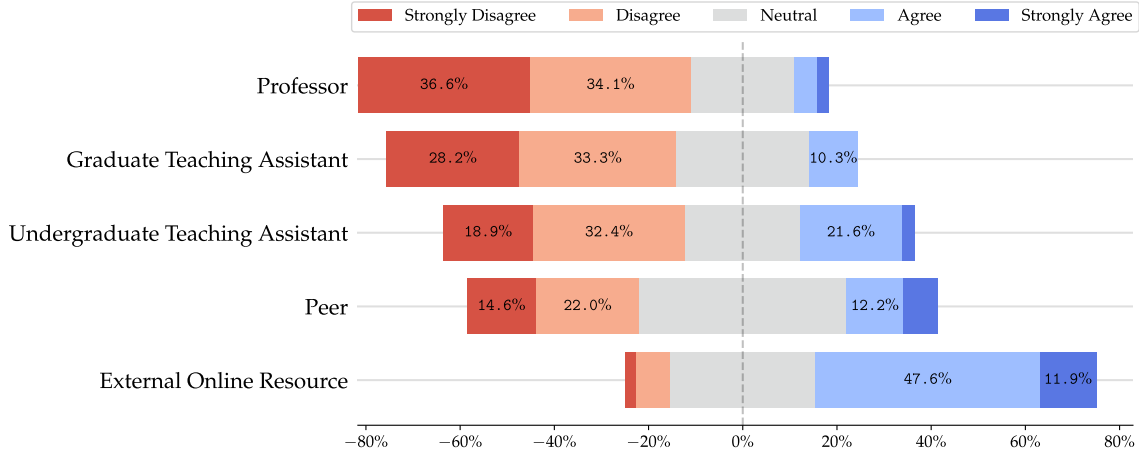
who opted to participate in the survey, 38% strongly agree that **SENSAI** is useful, 40% agree, 17% are neutral, and 5% disagree. Similarly, 45% strongly agree that **SENSAI** helps them learn, 38% agree, 10% are neutral, and 7% disagree. Figure 19 shows more broad learner perspective of **SENSAI**.

While sentiment is broadly very positive, learners are more divided about the speed of **SENSAI**'s response time. In our study, **SENSAI**'s response time to learners was between 5 seconds and 55 seconds, with a median of 18 seconds. Unlike ChatGPT, **SENSAI** does not stream its response word by word, but instead waits until it has a complete response to send. This design choice was to ensure that **SENSAI**'s "thinking" could be easily separated from its "response" (see Section 4.3). However, streaming the response word by word is possible, and parsing the response in-stream could significantly reduce the perceived response time.

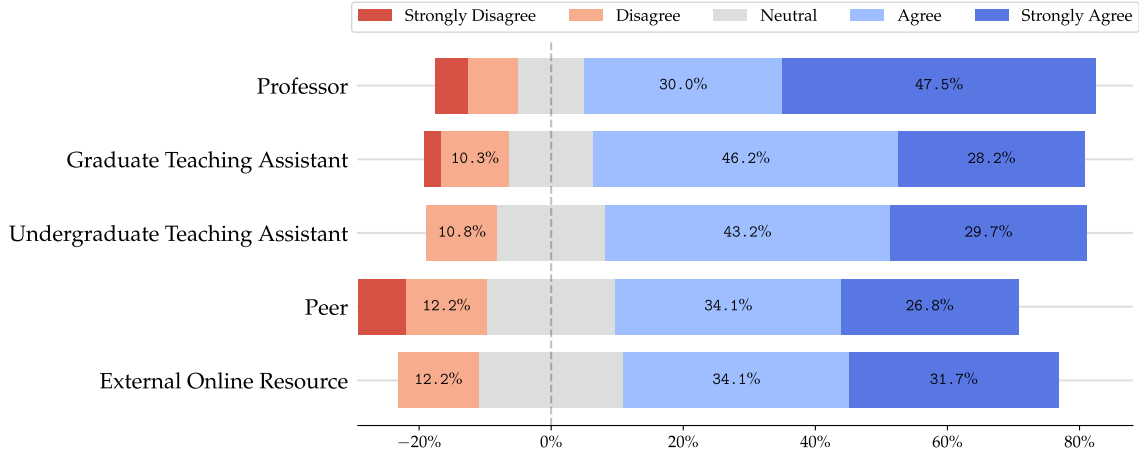
"It is a little slow but the wait is worth it."

Student Feedback

We further asked learners to rate the quality of **SENSAI**'s help on different topics, which is shown in Figure 20. Unsurprisingly, learners found the quality of **SENSAI** to be highest when seeking help with Python and Linux commands. Most of our learners have some Computer Science background, but do not necessarily have extensive experience with using Python or standard Linux command line operations. In such cases, **SENSAI** is able to provide significant syntactic assistance. On the other hand, learners found **SENSAI** to be less helpful when seeking help with Reverse Engineering and GDB. This is again unsurprising. Getting the relevant context to **SENSAI** for these topics is more difficult, due to large volumes of textual context and *graphical* context, which **SENSAI** cannot handle. Furthermore, these topics require a deeper understanding and methodology to be effective, as opposed to the more *reflexive intelligence* (see Section 4.7.4) that **SENSAI** shines at.



(a) “I prefer the quality of **SENSAI**’s guidance over a(n) ...’s guidance”.



(b) “It is easier to seek **SENSAI**’s guidance over a(n) ...’s guidance”.

Figure 21. Survey responses from 42 students regarding **SENSAI**’s guidance. (a) Quality comparison. (b) Accessibility comparison.

Finally, we asked learners to compare **SENSAI**’s guidance to that of a human tutor, both in terms of quality and accessibility. Unsurprisingly, learners found **SENSAI** to be more accessible than a human tutor, but less helpful in terms of quality. However, the difference in quality is not as large as might be expected, as shown in Figure 21, especially in the case of peers and undergraduate teaching assistants. **SENSAI** received overwhelmingly positive feedback when compared to external online resources.

"A knowledgeable human is still preferred, but that is difficult to scale."

Student Feedback

4.7 Analysis and Discussion

To ensure an in-depth understanding of **SENSAI**'s performance, we manually reviewed a significant subset of the 3,583 sessions, aiming to discern both its capabilities and potential areas for improvement. We read through these interactions between **SENSAI** and the learners, meticulously analyzing each message to understand the subtle nuances in the learners' questions and misunderstandings, how **SENSAI** responded, how the learners reacted, and how the sessions progressed—seeking to understand if **SENSAI** truly helped the learner educationally. To guide our review process, we first began with a random sample of the 3,583 sessions, and then iteratively selected additional sessions to further understand the themes that emerged by filtering for sessions that had particular characteristics, such as a specific challenge, time to solve, and semantics of the learner questions and **SENSAI**'s responses. We were able to filter for semantics by embedding the messages into a high-dimensional space (using an OpenAI embedding model [68]), and then using a nearest-neighbor search to find similar messages. This process was very useful as it allowed us to quickly identify sessions that had characteristics that we were interested in, and then manually review them to understand the nuances of the interactions. In total, we manually reviewed over 500 sessions, and through this process, we extract common themes that consistently emerged, reflecting on **SENSAI**'s, as well as LLMs', strengths and limitations. We discuss these themes in detail.

4.7.1 Importance of Context

The insight that stood out most prominently was the importance of context. SENSAT’s capability to access the learner’s context—specifically their recently active terminals and edited files—was essential for providing personalized and direct guidance. In instances where terminal context was absent, SENSAT frequently encountered difficulties in comprehending the nuances of the learner’s question and its relation to the challenge at hand. While SENSAT receives a basic description of the challenge (e.g., “manually send an Internet Protocol packet”), this brief explanation often omits crucial details vital for understanding the objective of the challenge—details that are typically revealed upon initially running the challenge. In cases where the learner clears their terminal, or begins discussing a new challenge without running it, SENSAT is unable to see this challenge output.

For example, in numerous modules, the goal is not necessarily to *exploit* a vulnerability, but rather to carry out an action that demonstrates comprehension of a fundamental concept; for example, sending a specially crafted packet, or writing a simple assembly program. These fundamental tasks are crucial for novice learners to tackle, as they lay the groundwork for more advanced, *exploitation*, tasks later on. However, when provided with limited context surrounding the challenge’s objective, SENSAT sometimes misguides the learner by asserting, for example, “you’re trying to exploit a vulnerability in the program to gain root access and read the contents of the ‘/flag’ file.” SENSAT assumes the existence of a vulnerability due to the *System Prompt* (refer to Figure 13), which indicates that the learner acts as the “hacker” user and must manipulate some “challenge” program to access the “flag”, despite not explicitly mentioning that a challenge will necessarily have some vulnerability to exploit. Given

limited context, this assumption is statistically probable, and is even correct for many challenges—but is also incorrect for many others, such as these fundamental tasks.

"It has provided good guidance on certain challenges by helping me understand the concept so that I could try to implement a solution. It is also good at giving me a starting point if I'm not sure where to begin."

Student Feedback

Unfortunately, crucial context might not be present in the challenge description, which is often a brief explanation of the challenge’s objective, but might instead be revealed upon initially running the challenge. If the learner does not run the challenge, clears their terminal, or otherwise obscures a copy of initial challenge output, **SENSAI** is unable to see this context, and so is only able to provide guidance based on the limited context it has. While learners who have some understanding of the challenge can discern such obvious misguidance and help guide **SENSAI** in the right direction (so it may then help them, as we observed), novice learners, sometimes unsure of how to even begin, may unfortunately be misled by this response, potentially leading to confusion and frustration.

While this “assuming” behavior of an LLM is a limitation here, it is also the source of **SENSAI**’s strength to provide useful guidance in many other cases. For example, seeing an error message in the terminal context, *correctly assuming* the associated line of code in the file context which caused the error, and *correctly assuming* the learner’s misunderstanding—therefore providing useful guidance on how to fix it. Therefore, our findings underscore the importance and tricky nature of shaping the context, and the necessity in order to provide accurate and educational guidance.

4.7.2 Unknown Unknowns

Learners often do not know what they do not know. One particularly interesting interaction involved the learner pasting part of a Python exception into their message to **SENSAI**, prompting **SENSAI** to guide them on what was going wrong. The learner, however, did not include the full traceback, and critically was missing the line that showed the actual error.

This is a common mistake for a novice, who may not understand what information is contextually important or not when diagnosing an issue. **SENSAI**, however, used the learner’s terminal to recover the remaining portion of the traceback, understand the issue, and guide the learner towards a solution. Even more interestingly, this was a case where **SENSAI** was able to understand the learner’s issue, but an experienced peer helping them in the online community (which we observed) was not, due to the subtle nature of the issue. This highlights the importance of **SENSAI**’s ability to “look over the learner’s shoulder” and see what they are working on in order to provide personalized and direct guidance.

4.7.3 Size of the Context Window

One of the most common technical issues we encountered was the learner’s terminal and file context overflowing the 8,192-token context window of GPT-4. This issue was particularly prevalent in the case of a large amount of terminal output. For example, in many modules, running the challenge itself produces a large amount of output, which the learner is able to view in the terminal. This output is very valuable context for **SENSAI** to understand both the objective of the challenge, and in some cases, what

the challenge reports back to the learner as a result of their actions. However, running the challenge multiple times, with each attempt, the learner’s terminal context grows, and eventually overflows the context window. There may be some value for SENSAT in being able to see the results of their past attempts, but much of this information may be highly redundant. Furthermore, in some cases, the learner may use additional tools such as `tcpdump` for network challenges, or the GNU debugger (`gdb`) for binary exploitation challenges, which produce a large amount of output. In its current state, SENSAT is entirely unable to handle this situation, other than to inform the learner that there is too much content, and to try again with less.

4.7.4 Reflexive Intelligence

SENSAT excels in understanding and responding to “complete context and knowledge”—context that elicits an immediate answer based on comprehensive knowledge. For instance, a conceptual question like “What is a buffer overflow?” or a syntactic query such as “How do I use the `curl` tool?” are examples of complete context—if the knowledge exists within the model, it can be promptly retrieved and presented to the learner. This capability extends far beyond direct questions; SENSAT can also interpret error messages, along with the learner’s terminal and file context, and provide immediate guidance on the issue, even if the learner does not recognize that their primary concern should be an error message, nor directly inquire about it. This powerful ability is similar to an experienced human tutor, who has encountered the same error message numerous times before and can instantly recognize it and offer advice. This *familiarity* that triggers an immediate and seemingly *reflexive* response

is a key strength of SENSAT, and is especially beneficial for novice learners, who have not yet developed this familiarity.

However, the ability to appropriately respond to “incomplete context or knowledge” is much more difficult for SENSAT. For example, a learner may ask “What is wrong with my approach?”. If the model recognizes the issue, it can provide immediate useful guidance. However, when it does not, SENSAT struggles.

An expert human tutor will also struggle in this situation, in comparison to reflexive familiarity. A good tutor, however, understands that the solution lies in rigorous research, creative experimentation, and thorough debugging, and, in fact, uses the experience as a teachable moment for the learner. They might use tools to delve into the inner workings of a program at a particular state or watch the exchange of network packets between systems. An expert human tutor knows how to form hypotheses and test them. Even better, they can guide the learner through this process, helping them to form hypotheses and test them, and fostering a culture of inquiry by guiding them to harness these tools themselves. This is arguably the most important skill that a tutor can teach a cybersecurity student: how to debug a problem.

However, just as budding learners may grapple with the nuances of debugging or occasionally even show an aversion to it, SENSAT too faces similar limitations. While SENSAT may in some cases encourage the learner to try a specific tool or approach, it is often unable to provide direct and accurate guidance on how to approach the task of forming useful hypotheses and accurately testing them. While the learner may be able to use SENSAT’s suggestion to, for example, run the program in a debugger, and SENSAT may reflexively notice a specific issue during this process, it nevertheless relies on this reflexive capability. Like the learner, SENSAT struggles with the unknown unknowns—but unfortunately, speaks with authority even when it is highly uncertain.

This distinction is underscored by its capacity to conceptually understand and explain challenges. For example, **SENSAI** can coherently explain the nature and implications of security mechanisms such as PIE and stack canaries, or vulnerabilities such as buffer overflows and SQL injections. It even possesses the descriptive knowledge to convincingly advise on overcoming these mechanisms in environments such as a pwntools script. However, the application of this knowledge in practice is where the model often falters. This dichotomy aligns with the observations made by Ma et al. [50], emphasizing that while GPT excels in grasping code syntax and certain static behaviors, it struggles in dynamic environments. Nevertheless, **SENSAI**'s ability to *reflexively* address a wide range of questions, confusions, and errors is a powerful capability that can be leveraged to enhance the learning experience.

*"I think the biggest benefit of using **SENSAI** is eliminating the chore of parsing search results and documentation to learn syntax."*

Student Feedback

4.7.5 Hallucinations

Hallucinations is the term that describes when an LLM responds with information that is incorrect or fictional (which is not surprising given the statistical nature of the model). We found that learners should be taught about the boundaries and limitations of LLMs. One example includes **SENSAI** informing a learner, "You're using syscall numbers from regular Linux, however the learning environment uses a simplified syscall interface. As a hint, the syscall number for `sys_open` in the learning platform is 5." While it's certainly possible that a learning environment could use a simplified syscall interface, this is not the case in our environment. This is completely fabricated,

and accentuates the importance of leveraging **SENSAI** within its recognized strengths and verifying its suggestions.

4.8 Conclusion

We developed **SENSAI**, an AI-powered tutoring system that leverages modern Large Language Models (LLMs) to provide personalized feedback and guidance to learners. **SENSAI**'s novel design automatically extracts the learner's context—their recently active terminals and edited files—and we show that access to this context is *crucial* for **SENSAI**'s performance. We analyze the strengths and limitations of **SENSAI**, noting the limitations of the context window and the LLM's “reflexive intelligence”—it's ability to excel at providing baked-in knowledge, but struggle to debug complex problems. We hope that our study will serve as a foundation for future explorations in personalized large language model (LLM) tutors, and shift the ratio of students to educators back in favor of the students.

Chapter 5

THE PWN.COLLEGE

I present *the pwn.college*, which is the culmination of the PWN methodology, DOJO infrastructure, automated SENSAT tutoring, and many more incredible innovations—and is available for free to the world at <https://pwn.college>. Here, I discuss the current state of *pwn.college*, and present a collection of statistics surrounding its usage and impact on the cybersecurity education community, as of the time of writing in February 2024.

pwn.college was created in response to the immediate needs for a systems security course, CSE 466, at Arizona State University (ASU) in the Fall of 2018. *pwn.college* was released as a public beta in 2019, and left beta in 2021. In 2022, *pwn.college* expanded to include topics and infrastructure needs for CSE 365, ASU’s introduction to cybersecurity course, which all Computer Science students are required to take.

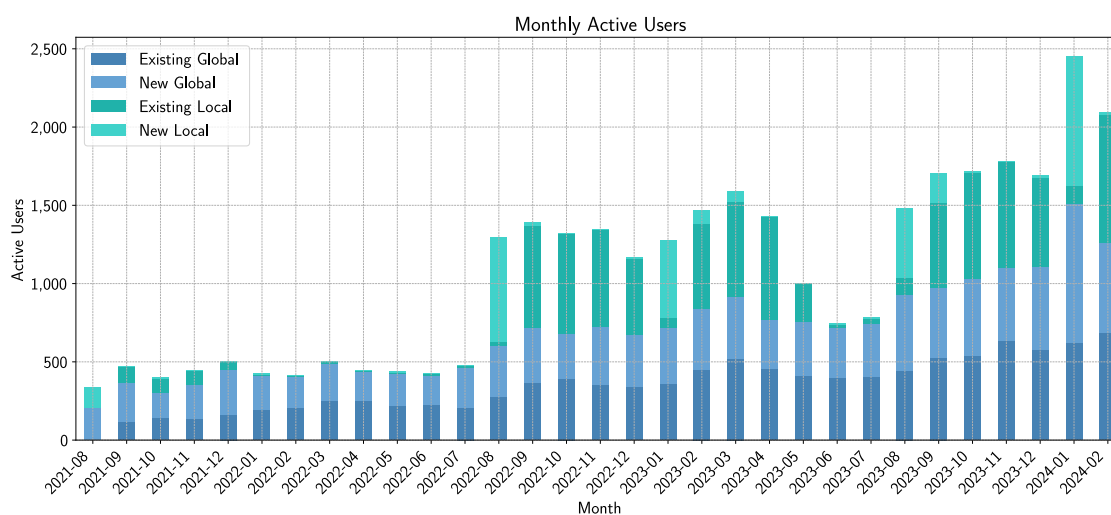


Figure 22. Monthly active users.

As shown in Figure 22, `pwn.college` has seen a steady increase in monthly active users, with approximately 300 monthly active users in August 2021, and now approximately 2,500 monthly active users as of January 2024. Unsurprisingly, we experienced significant growth in August 2022 with the release of CSE 365 content.

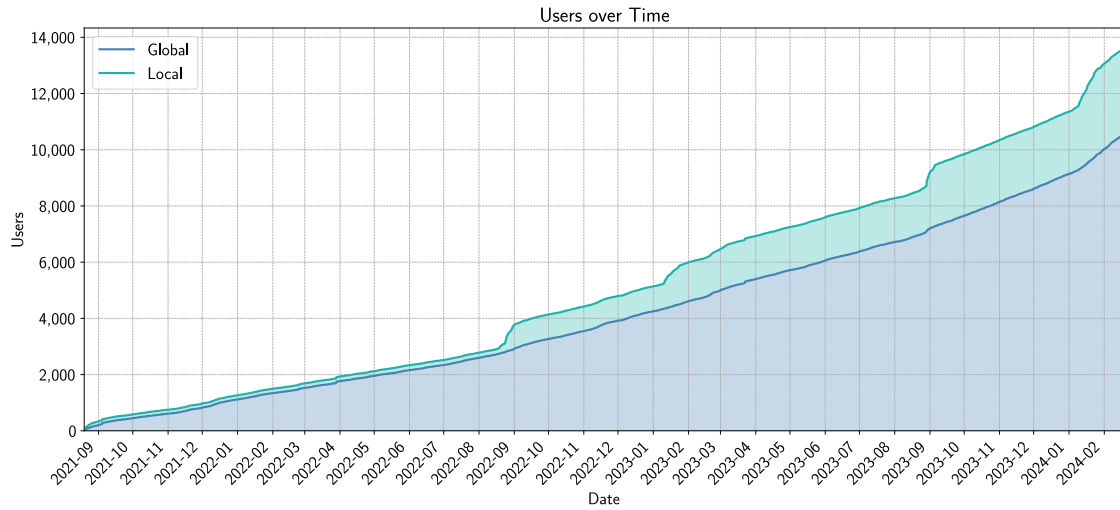


Figure 23. Users over time.

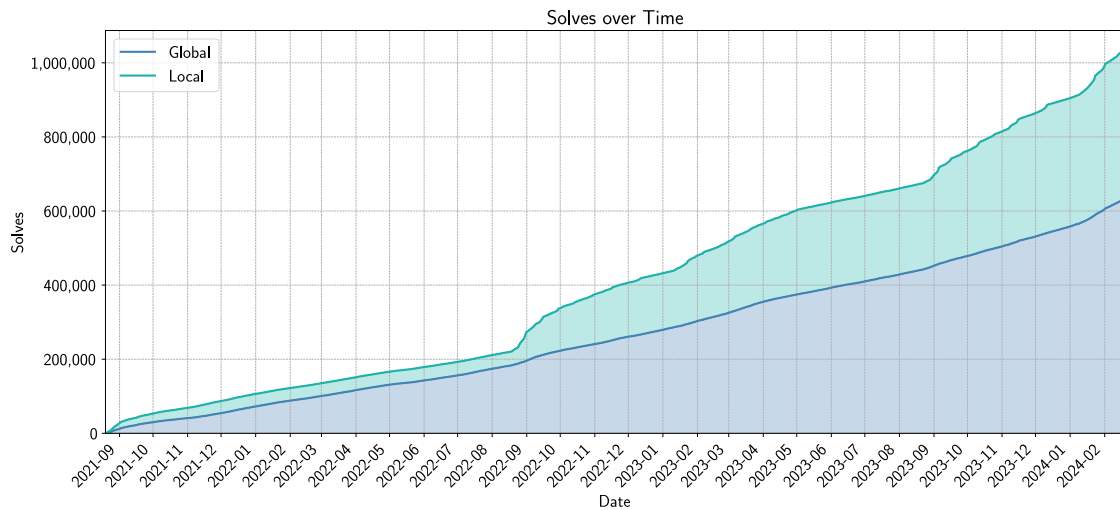


Figure 24. Solves over time.

To date, we have seen over 13,000 unique users, with the vast majority of those users being “global” (non-ASU) students, as seen in Figure 23. In February 2024, we reached over 1,000,000 solves, as seen in Figure 24. While the vast majority of our users are “global”, a more significant (but still less than half) portion of our solves come from “local” (ASU) students. This makes sense, as these students are required to solve challenges for a grade!

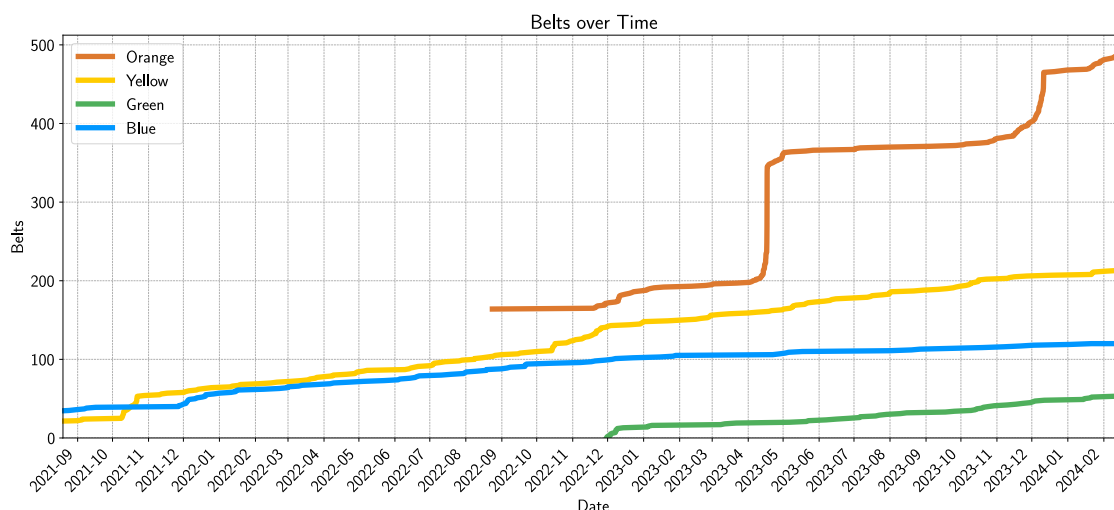


Figure 25. Belts over time.

Hundreds of users have earned belts, which are awarded for solving complete sets of challenges, as seen in Figure 25. In the beginning, we only had yellow belts. We then added blue belts, and then eventually orange belts, and finally green belts are the most recent addition. While the meaning of these belts have evolved with time, they have always been a way to recognize and reward students for their hard work and dedication to learning cybersecurity, and we have shipped physical belts to students who have earned them. As of writing, in order of difficulty, we currently award the orange belt for completing the “Introduction to Cybersecurity” topic, yellow belt for completing the “Program Security” topic, green belt for completing the “System

Security” topic, and blue belt for completing the “Software Exploitation” topic. These belts are now cumulative (i.e., you must have all previous belts to earn a new belt), but were not always (which explains the non-monotonicity of Figure 25). With time, we may continue to add more belts, or adjust the requirements for existing belts, as we continue to expand the `pwn.college` platform. We hope that one day, we might have a full curriculum of belts that expands all the way to the black belt, and that anyone with enough dedication and hard work can achieve this level of expertise, *for free*.

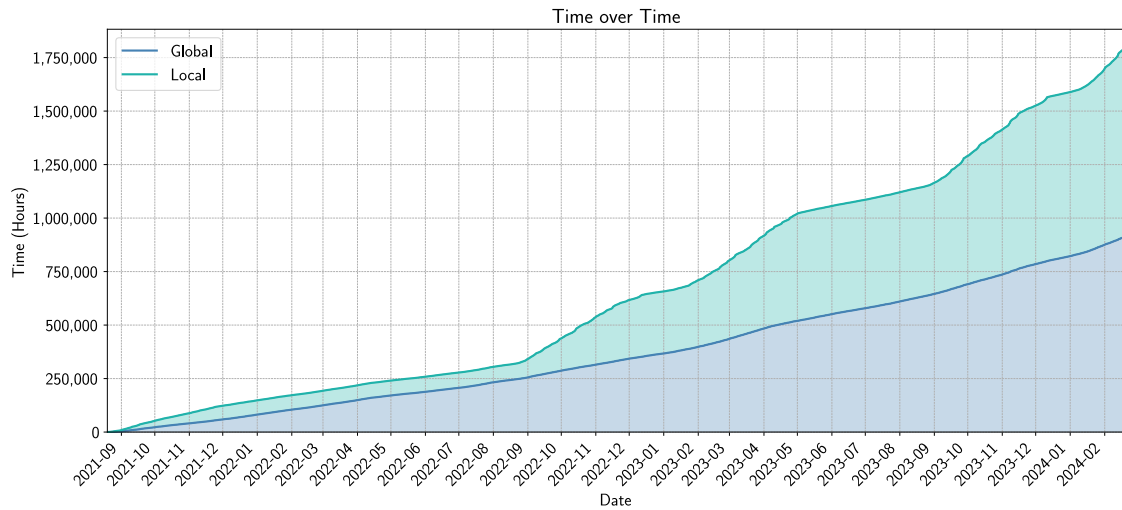


Figure 26. Time spent hacking over time.

Unfortunately we do not have a good way to measure the total time spent hacking on `pwn.college`. However, we can estimate this by computing differences in flag submission times, and carefully eliminating outlier times (for example, as a result of going to sleep and resuming the next day). In doing so, we estimate that over 1,750,000 hours (over 200 years) have been spent hacking on `pwn.college`, as seen in Figure 26. Due to the uncertainty in this estimate, this number might be closer to 1,000,000 hours, or maybe even closer to 3,000,000 hours. Regardless, this is a

significant amount of time, and we are proud to have been able to provide this platform for so many students to learn cybersecurity.

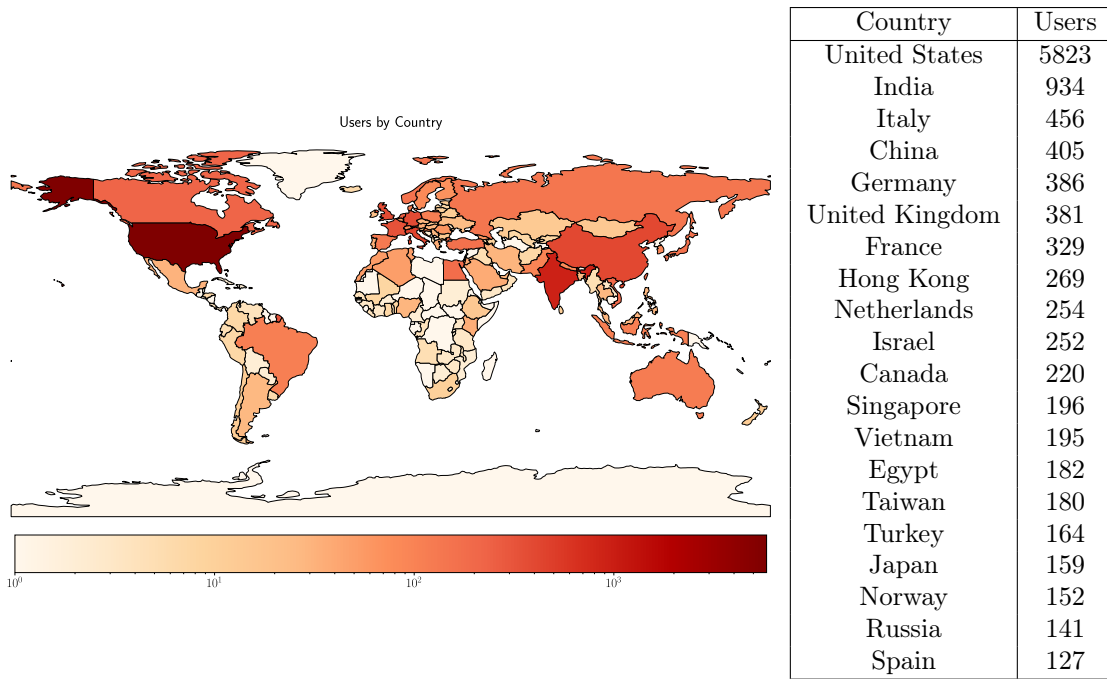


Figure 27. Users by country. In total, we have users from 145 countries.

We have been able to reach students from all around the world, as seen in Figure 27. In total, we have seen users from 145 countries. While there is a long tail of countries with only a few users, and it is unclear if these users are actually from these countries (VPNs make this difficult to determine), we have nevertheless been able to impact the broader global community of cybersecurity students, with the United States accounting for the largest portion of our users, but more than half of our users coming from outside the United States.

I am proud of the impact that `pwn.college` has had on the cybersecurity education community, and am excited to continue to expand and improve the platform in order to continue to provide the best possible cybersecurity education to the world, at scale, *for free*.

Chapter 6

THE FUTURE OF CYBERSECURITY EDUCATION

We certainly were not the first to innovate the idea of hacking as a way to learn. In fact, I would argue that the truest essence of hacking *is* the pursuit of knowledge. In this work, we have tried to capture that essence, and to bring it to the masses—where it belongs.

So what is next? *Lots*. We have only scratched the surface of what is possible in cybersecurity education, and I am excited to see what the future holds. One of the incredible opportunities that the combined `pwn.college` platform—the PWN methodology, DOJO infrastructure, and automated SENSEI tutoring—offers us is the ability to collect data at scale (*thousands* of users) and in-depth (every *keystroke* and every *misunderstanding*). This data can be used to drive new insights into how people learn, and to develop new tools to help them learn better: we now have the base for an incredible *cybersecurity education research platform*.

As an instructor, when I ask a question during a lecture, I am often met with blank stares and silence. Sometimes a few students will raise their hands, but often they are not the ones who need the most help. I have found that the students who need the most help are often the ones who are the most unwilling to ask for it (until maybe a few hours before the assignment is due). This is a problem that SENSEI can help solve—not just for the student, but for the instructor as well.

When a student asks a question, SENSEI can provide an answer. Hopefully that help is good (correct and educational), and we have found that it often is. But possibly even more important, SENSEI can provide the instructor with a window into

the student's mind. That student that won't ask a question in class, but will ask SENSAT, is now asking *you* a question. This is a powerful tool for an instructor, and it is one that I am only beginning to understand how to use. This level of insight synergizes well with the PWN methodology, enabling us to see where we need to create new challenges and refine existing to further smoothen the learning curve, and the DOJO infrastructure which allows us to create and deploy those challenges in a matter of minutes, with new resources and tooling immediately available to all. One student's misunderstanding is now an improved learning experience for *everyone*.

With such a data-rich feedback loop, the future of cybersecurity education is bright. We can begin to truly understand what is holding students back, and how to help them move forward. I think in the future, someone will look back at this work and wonder what the heck we were doing, teaching with such little insight into the student's mind. I hope that this work is a step in the right direction of taking education to the next level—in taking *hacking* to the next level.

REFERENCES

- [1] Ethan Ake-Little, Nathaniel von der Embse, and Dana Dawson. “Does class size matter in the university setting?” In: *Educational Researcher* 49.8 (2020), pp. 595–605.
- [2] Apple. *Apple Security Bounty Categories*. 2023. URL: <https://security.apple.com/bounty/categories/>.
- [3] John Aycock et al. “Exercises for Teaching Reverse Engineering”. In: *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. 2018, pp. 188–193.
- [4] Rishabh Balse et al. “Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments”. In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 2023, pp. 292–298.
- [5] Masooda Bashir et al. “An Examination of the Vocational and Psychological Characteristics of Cybersecurity Competition Participants”. In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.
- [6] Nada Basit et al. “A Learning Platform for SQL Injection”. In: *Proceedings of the 50th ACM technical symposium on computer science education*. 2019, pp. 184–190.
- [7] Fabrice Bellard. “QEMU, a Fast and Portable Dynamic Translator”. In: *USENIX annual technical conference, FREENIX Track*. Vol. 41. 46. California, USA. 2005, pp. 10–5555.
- [8] Razvan Beuran et al. “Integrated framework for hands-on cybersecurity training: CyTrONE”. In: *Computers & Security* 78 (2018), pp. 43–59.
- [9] Hack The Box. <https://www.hackthebox.com/>. 2023.
- [10] Root The Box. <https://github.com/moloch--/RootTheBox>. 2023.
- [11] Sergey Bratus, Anna Shubina, and Michael E Locasto. “Teaching the Principles of the Hacker Curriculum to Undergraduates”. In: *Proceedings of the 41st ACM technical symposium on computer science education*. 2010, pp. 122–126.

- [12] Clarissa Brierley, Leila Ellis, and Emily Roisin Reid. “Peer-assisted learning in medical education: a systematic review and meta-analysis”. In: *Medical Education* 56.4 (2022), pp. 365–373.
- [13] Jonathan Burket et al. “Automatic Problem Generation for Capture-the-Flag Competitions”. In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.
- [14] Martin Carlisle, Michael Chiaramonte, and David Caswell. “Using CTFs for an Undergraduate Cyber Education”. In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.
- [15] Peter Chapman, Jonathan Burket, and David Brumley. “PicoCTF: A Game-Based Computer Security Competition for High School Students”. In: *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. 2014.
- [16] Kevin Chung. “Live Lesson: Lowering the Barriers to Capture The Flag Administration and Participation”. In: *2017 USENIX Workshop on Advances in Security Education (ASE 17)*. 2017.
- [17] Kevin Chung and Julian Cohen. “Learning Obstacles in the Capture The Flag Model”. In: *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. 2014.
- [18] VS Code. <https://code.visualstudio.com/>. 2023.
- [19] DEF CON Hacking Conference. <https://defcon.org/>. 2023.
- [20] DEF CON Hacking Conference. *DEF CON CTF*. <https://defcon.org/html/links/dc-ctf.html>. 2024.
- [21] DEF CON Hacking Conference. *Packet Hacking Village*. <https://www.wallofsheep.com/pages/packet-hacking-village>. 2024.
- [22] DEF CON Hacking Conference. *warl0ck gam3z CTF*. <https://warl0ck.gam3z.com>. 2024.
- [23] William Crumpler and James A Lewis. *The cybersecurity workforce gap*. JSTOR, 2019.
- [24] CTFtime. <https://ctftime.org/>. 2023.
- [25] CTFtime. *CTF WTF*. <https://ctftime.org/ctf-wtf/>. 2024.

- [26] Adrian Dabrowski et al. “Leveraging Competitive Gamification for Sustainable Fun and Profit in Security Education”. In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.
- [27] DEFKOR. *DEFKOR CTF Team*. <https://ctftime.org/event/213/>. 2024.
- [28] Michael H Dunn and Laurence D Merkle. “Assessing the Impact of a National Cybersecurity Competition on Students’ Career Interests”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 2018, pp. 62–67.
- [29] Joshua Eckroth et al. “Alpaca: Building Dynamic Cyber Ranges with Procedurally-Generated Vulnerability Lattices”. In: *Proceedings of the 2019 ACM Southeast Conference*. 2019, pp. 78–85.
- [30] Facebook. <https://github.com/facebookarchive/fbctf>. 2023.
- [31] Wu-chang Feng. “A Scaffolded, Metamorphic CTF for Reverse Engineering”. In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.
- [32] Efstratios Gavvas, Nasir Memon, and Douglas Britton. “Winning Cybersecurity One Challenge at a Time”. In: *IEEE Security & Privacy* 10.4 (2012), pp. 75–79.
- [33] GitHub. *GitHub Feature: Codespaces*. 2021. URL: <https://github.com/features/codespaces/>.
- [34] GitHub. *GitHub’s Engineering Team has moved to Codespaces*. 2021. URL: <https://github.blog/2021-08-11-githubs-engineering-team-moved-codespaces/>.
- [35] Google. <https://github.com/google/kctf>. 2023.
- [36] Arto Hellas et al. “Exploring the Responses of Large Language Models to Beginner Programmers’ Help Requests”. In: *arXiv preprint arXiv:2306.05715* (2023).
- [37] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. 2021. arXiv: 2009.03300 [cs.CY].
- [38] Erfan Al-Hossami et al. “Socratic Questioning of Novice Debuggers: A Benchmark Dataset and Preliminary Evaluations”. In: *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*. 2023, pp. 709–726.

- [39] ISACA. *State of Cybersecurity 2022: Global Update on Workforce Efforts, Resources and Cyberoperations*. Tech. rep. ISACA, 2022.
- [40] Jinja. <https://jinja.palletsprojects.com/>. 2023.
- [41] Samy Kamkar. *Drive it like you Hacked it: New Attacks and Tools to Wirelessly Steal Cars*. Presented at DEF CON 23. 2015.
- [42] Stylianos Karagiannis et al. “Sandboxing the Cyberspace for Cybersecurity Education and Learning”. In: *European Symposium on Research in Computer Security*. Springer. 2020, pp. 181–196.
- [43] Salah Kassab et al. “Student-led tutorials in problem-based learning: educational outcomes and students’ perceptions”. In: *Medical teacher* 27.6 (2005), pp. 521–526.
- [44] Jean-François Lalande et al. “Teaching Android Mobile Security”. In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 2019, pp. 232–238.
- [45] Sam Lau and Philip J Guo. “From " Ban It Till We Understand It" to" Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot”. In: (2023).
- [46] Juho Leinonen et al. “Using large language models to enhance programming error messages”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 2023, pp. 563–569.
- [47] Kees Leune and Salvatore J Petrilli Jr. “Using Capture-the-Flag to Enhance the Effectiveness of Cybersecurity Education”. In: *Proceedings of the 18th Annual Conference on Information Technology Education*. 2017, pp. 47–52.
- [48] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.
- [49] Rongxin Liu et al. “Teaching CS50 with AI”. In: (2024).
- [50] Wei Ma et al. *The Scope of ChatGPT in Software Engineering: A Thorough Investigation*. 2023. arXiv: 2305.12138 [cs.SE].

- [51] Jakub Macina et al. “MathDial: A Dialogue Tutoring Dataset with Rich Pedagogical Properties Grounded in Math Reasoning Problems”. In: *arXiv preprint arXiv:2305.14536* (2023).
- [52] David J Malan. “Standardizing Students’ Programming Environments with Docker Containers: Using Visual Studio Code in the Cloud with GitHub Codespaces”. In: *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2*. 2022, pp. 599–600.
- [53] David J Malan et al. “Providing Students with Standardized, Cloud-Based Programming Environments at Term’s Start (for Free)”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*. 2022, pp. 1183–1183.
- [54] Julia M Markel et al. “GPTeach: Interactive TA Training with GPT Based Students”. In: (2023).
- [55] Meta. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].
- [56] Charlie Miller and Chris Valasek. *Remote Exploitation of an Unaltered Passenger Vehicle*. Presented at DEF CON 23. 2015.
- [57] Jelena Mirkovic and Peter AH Peterson. “Class Capture-the-Flag Exercises”. In: *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. 2014.
- [58] Jelena Mirkovic et al. “Engaging Novices in Cybersecurity Competitions: A Vision and Lessons Learned at ACM Tapia 2015”. In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.
- [59] Jelena Mirkovic et al. “Using Terminal Histories to Monitor Student Progress on Hands-on Exercises”. In: *Proceedings of the 51st ACM Technical Symposium on Computing Science Education*. 2020, pp. 866–872.
- [60] Moodle. <https://moodle.com/>. 2023.
- [61] Connor Nelson and Yan Shoshitaishvili. “DOJO: Applied Cybersecurity Education In The Browser”. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE)*. Mar. 2024.

- [62] Connor Nelson and Yan Shoshitaishvili. “PWN The Learning Curve: Education-First CTF Challenges”. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE)*. Mar. 2024.
- [63] Mike O’Leary. “Innovative Pedagogical Approaches to a Capstone Laboratory Course in Cyber Operations”. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 2017, pp. 429–434.
- [64] TJ O’Connor, Ben Sangster, and Erik Dean. “Using Hacking to Teach Computer Science Fundamentals”. In: *American Society for Engineering Education, St. Lawrence Section* (2010).
- [65] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].
- [66] OpenAI. *Prompt Engineering: Six strategies for getting better results*. <https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results>. 2023.
- [67] OpenAI. *Strategy: Give GPTs time to "think"*. 2024. URL: <https://platform.openai.com/docs/guides/prompt-engineering/give-the-model-time-to-think>.
- [68] OpenAI. *What are embeddings?* 2024. URL: <https://platform.openai.com/docs/guides/embeddings/what-are-embeddings>.
- [69] Order Of The Overflow. <https://archive.ooo/>. 2023.
- [70] PapersWithCode. *SOTA for Multi-task Language Understanding on MMLU*. Access date: 2023-08-16. 2023. URL: <https://paperswithcode.com/sota/multi-task-language-understanding-on-mmlu>.
- [71] Younghee Park et al. “Enhancing Security Education Through Designing SDN Security Labs in CloudLab”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 2018, pp. 185–190.
- [72] Tung Phung et al. “Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors”. In: *International Journal of Management* 21.2 (2023), p. 100790.
- [73] KYPO Project. <https://gitlab.ics.muni.cz/groups/muni-kypo-crp/-/epics/25>. 2022.
- [74] pwnable.kr. <https://pwnable.kr/>. 2023.
- [75] pwnable.tw. <https://pwnable.tw/>. 2023.

- [76] Plaid Parliament of Pwning. *Plaid Parliament of Pwning CTF Team*. <https://pwning.net/>. 2024.
- [77] Ronald Reagan. *Tear Down This Wall*. Speech presented at the Brandenburg Gate. Berlin, Germany, June 1987.
- [78] ret2. <https://wargames.ret2.systems/>. 2023.
- [79] Jaromir Savelka et al. “Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses”. In: *arXiv preprint arXiv:2306.10073* (2023).
- [80] Z Cliffe Schreuders et al. “Security Scenario Generator (SecGen): A Framework for Generating Randomly Vulnerable Rich-scenario VMs for Learning Computer Security and Hosting CTF Events”. In: *2017 USENIX Workshop on Advances in Security Education (ASE 17)*. 2017.
- [81] SecDevOps@Cuse. <https://github.com/secdevops-cuse/CyberRange>. 2023.
- [82] Shellphish. *Shellphish CTF Team*. <https://shellphish.net/>. 2024.
- [83] Yan Shoshitaishvili and Ruoyu Wang. *Angry Hacking: The next gen of binary analysis*. Presented at DEF CON 23. 2015.
- [84] Frank J Sinkavich. “Performance and metamemory: Do students know what they don’t know?” In: *Journal of Instructional psychology* 22.1 (1995), p. 77.
- [85] Xin Tan. “The Impact of ChatGPT on Education and Future Prospects”. In: *Highlights in Science, Engineering and Technology* 61 (2023), pp. 138–143.
- [86] Gemini Team et al. “Gemini: a family of highly capable multimodal models”. In: *arXiv preprint arXiv:2312.11805* (2023).
- [87] Haoye Tian et al. “Is ChatGPT the Ultimate Programming Assistant—How far is it?” In: *arXiv preprint arXiv:2304.11938* (2023).
- [88] TryHackMe. <https://tryhackme.com/>. 2023.
- [89] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
- [90] Giovanni Vigna et al. “Ten Years of iCTF: The Good, The Bad, and The Ugly”. In: *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. 2014.

- [91] Jan Vykopal, Valdemar Švábenskỳ, and Ee-Chien Chang. “Benefits and Pitfalls of Using Capture the Flag Games in University Courses”. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 2020, pp. 752–758.
- [92] Jan Vykopal et al. “KYPO Cyber Range: Design and Use Cases”. In: *Proceedings of the 12th International Conference on Software Technologies - Volume 1: ICSOFT*. SciTePress. 2017, pp. 310–321.
- [93] Jason Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 24824–24837. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.
- [94] Richard Weiss, Michael E Locasto, and Jens Mache. “A Reflective Approach to Assessing Student Performance in Cybersecurity Exercises”. In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 2016, pp. 597–602.
- [95] Richard S Weiss et al. “Teaching Cybersecurity Analysis Skills in the Cloud”. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 2015, pp. 332–337.
- [96] Joseph Werther et al. “Experiences In Cyber Security Education: MIT Lincoln Laboratory Capture-the-Flag Exercise”. In: *4th Workshop on Cyber Security Experimentation and Test (CSET 11)*. 2011.
- [97] Over The Wire. <https://overthewire.org/wargames/>. 2023.
- [98] Google Project Zero. *A deep dive into an NSO zero-click iMessage exploit: Remote Code Execution*. 2021. URL: <https://googleprojectzero.blogspot.com/2021/12/a-deep-dive-into-nso-zero-click.html>.
- [99] Lianmin Zheng et al. “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena”. In: *arXiv preprint arXiv:2306.05685* (2023).
- [100] Barry J Zimmerman and Manuel Martinez-Pons. “Development of a structured interview for assessing student use of self-regulated learning strategies”. In: *American educational research journal* 23.4 (1986), pp. 614–628.