Guided Learning and Interactive Visualization for Teaching & Learning Stack Smashing Attacks & Defenses: Experiences and Evaluation

Harini Ramaprasad
hramapra@charlotte.edu
UNC Charlotte
Charlotte, North Carolina, USA

Meera Sridhar
msridhar@charlotte.edu
UNC Charlotte
Charlotte, North Carolina, USA

Sushma I Dangeti
sdangeti@charlotte.edu
UNC Charlotte
Charlotte, North Carolina, USA

Soham Pradhan
spradh11@charlotte.edu
UNC Charlotte
Charlotte, North Carolina, USA

Islam Obaidat*
iaobaidat@ncat.edu
NC Agricultural and Technical State University
Greensboro, North Carolina, USA

***Abstract*—This Innovative Practice paper presents the design, deployment, and evaluation of a software security module that teaches stack smashing attacks and defenses using innovative pedagogical practices. Widely ubiquitous buffer overflow vulnerabilities and stack smashing attacks that exploit them are critical components in advanced software security curricula, since buffer overflows can arise due to simple programmer oversight, and stack smashing can have dangerous consequences in critical systems. However, these topics are known to be difficult to teach and learn due to the vast amount of background needed, the difficulty of learning type-unsafe languages, and laborious memory address space calculations involved. In this work, we aim to bring innovative pedagogical practices to this advanced cybersecurity education topic through a suite of four guided learning activities that follow the Process Oriented Guided Inquiry Learning (POGIL) style, and DISSAV, an interactive visualization tool for modeling stack smashing attacks. This paper presents an evaluation of the module based on deploying it in multiple sections of an introductory undergraduate cybersecurity course in the UNC Charlotte in Fall 2022, Spring 2023, and Fall 2023. Our study finds that students have mostly positive perceptions about activity structure / design, content, and style, but that improvements may be needed to some aspects, including question phrasing, activity length, and teamwork facilitation.**

***Index Terms*—guided learning, stack smashing, engaged pedagogy, software security, descriptive statistics**

I. Introduction

Buffer overflows and *stack smashing* attacks [18]—attacks that exploit buffer overflows to hijack the control-flow of a program—are staples in advanced software security curricula. Buffer overflows can be widespread due to simple programmer oversight, and stack smashing attacks that exploit them can be painfully dangerous [4]. One main challenge with teaching these critical software security topics is the amount of background knowledge needed (e.g., process memory layout, call stacks, shellcode, etc.), the difficulty of learning type unsafe languages such as C (since they provide fertile ground for buffer overflow vulnerabilities), and laborious memory address space calculations involved [3], [10], [18], [22], [23].

In this paper, we present the design, deployment, and evaluation of a software security module that teaches stack smashing attacks and defenses. Our module comprises (1) a suite of four **guided learning activities**, namely: (a) *Buffer Overflows in C*, (b) *Process Memory Layout*, (c) *Stack Smashing*, and (d) *Defenses*; and (2) an **active learning exercise** that uses an interactive visualization tool — **DISSAV** [1] — that we developed in prior work for modeling stack smashing attacks.

Our four guided learning activities follow the *Process Oriented Guided Inquiry Learning* or POGIL [21] style. The idea of POGIL is to teach students targeted concepts through collaborative activities that serve as students' first exposure to the concepts, i.e., no prior lecture / material is provided. POGIL-style activities also explicitly target the development of *process skills* (e.g., communication, teamwork, problem solving, etc.). The activities are designed to be used with *self-managed teams* of students and use a Guided Inquiry Learning cycle (or simply Learning Cycle). Specifically, they present learning models (e.g., program code, graphical representations of concepts, tables of data, etc.). Students *explore* the models via direct questions, then answer questions through which they *invent* concepts and may be introduced to new terms, and finally answer questions that require them to *apply* the concepts, before exploring more complex models / concepts.

POGIL-style activities have been shown to improve student mastery of content in a variety of areas and topics [20], and yet, to the best of our knowledge, there are currently

*Work done as a student at UNC Charlotte.

no POGIL-endorsed activities for advanced cybersecurity topics [5]. Our hypothesis is that the guided learning style of activities is well-suited to the topic of stack smashing due to the vast background information involved, to enable students to discover and build on concepts, gradually understanding more complex ones. Our active learning exercise guides students through the usage of DISSAV to create a C program with a buffer overflow vulnerability, construct a stack smashing attack payload, and execute the program to visualize the attack.

In this paper, we aim to answer the following research questions to investigate our hypothesis: **(R1)** Do students think that the guided learning activities and DISSAV are well designed and help them learn about stack smashing? **(R2)** Do students think that the guided learning activities and DISSAV are engaging? To answer these questions, we conducted an IRB-approved user study. We deployed our module and a student experience survey in an undergraduate, introductory cybersecurity course at our institution in Fall 2022 (2 sections, 89 students), Spring 2023 (1 section, 32 students), and Fall 2023 (2 sections, 99 students). We evaluate the effectiveness of the activities through a systematic analysis of survey data.

II. Literature Review

Several prior works discuss various strategies for teaching memory safety, buffer overflows and stack smashing attacks [6], [13], [23], [25], [27], [30]. For instance, Jones et al. [13] discuss a course module to teach students the impact of input flaws and buffer overflows on a program by using simple examples. Walker et al. [25] present a program analysis and visualization tool to help students understand the impact of common memory errors, such as buffer overflows, in C programs. Zhang et al. [30] present a web-based interactive visualization tool to teach buffer overflow concepts. The tool offers six components for progressive learning, including assessments for immediate feedback and a fun mini-game for added engagement. Unlike these works, our paper focuses on employing the guided learning paradigm.

Yuan et al. [29] present *Guided-Inquiry Collaborative Learning* (GICL), in which activities are developed based on Bloom's Taxonomy [15] verbs "analyze", "apply" and "critically think". The authors present GICL activities for network security, including briefly touching upon buffer overflows [29], and discuss GICL activities for teaching cybersecurity, including secure coding, in an online setting [9]. In an extended abstract, He et al. [8] discuss a GICL activity on buffer overflows. The authors mention that they have the results of a comparison of pre- and post-evaluation of student learning of deployment; however, the results are not discussed in the abstract. While exploring some form of guided learning-based pedagogy for teaching cybersecurity concepts, the above research does not focus on in-depth activities on stack smashing attack and defense education, as our work does. Phuong et al. [19] develop a Project-Guided Learning (PGL) framework to enhance cybersecurity education. They aim to equip students with practical skills in cryptography, access control, network security, and secure coding, supplemented with pre and post surveys. Contrary to these works, our activities utilize the POGIL style, fostering a more dynamic and engaging learning experience while enhancing comprehension and retention in the context of teaching stack smashing attacks and defenses.

POGIL style activities have recently been explored in cybersecurity education. Alshaher et al. [2] explore the implementation of POGIL to teach about flooding attacks, on the Software Defined Network (SDN) data plane. The activities aim to enhance student's understanding of flooding attacks, their causes, and potential defenses. Yang et al. [28] present their experiences with using POGIL to teach students about access control, including results from student surveys on a hands-on lab activity on access control. Unlike these works, the activities in this paper are designed to teach students foundational concepts leading up to and including stack smashing attacks and defenses. Tian and Li [24] conduct a pilot study on collaborative guided learning for information security topics, developing POGIL activities on input validation, security in operating systems, and SQL injection. Their goal is to determine student attitudes towards this pedagogy, factors influencing interest in cybersecurity, and helpful learning elements. The paper focuses primarily on outcomes of the activities rather than details of the study like our paper does.

III. Software Security Module

In this section, we briefly introduce a warm-up resource that presents prerequisite information, our suite of guided learning activities, and an active learning exercise that uses our interactive visualization tool. *Information Processing* and *Critical Thinking* are the two process skills that students will develop through our guided learning activities. Students work in groups of two to three and we suggest that they assign and use the roles of (1) Manager / Facilitator, (2) Timekeeper, and (3) Recorder / presenter among themselves We suggest that the groups rotate roles for each activity.

A. Warm-Up Resource: Strings in C

We create a warm-up resource with content about how C-style strings (hereafter referred to as "strings") are created, used and stored. We present the two options that can be used to define string variables in C—as an array of characters and as a pointer to a character. We provide multiple examples for each option, illustrating how data is stored in memory. We also introduce the string terminating *null* character (i.e., `'\0'`) and explain why it is used in C.

B. Buffer Overflows in C

Upon completing this activity, students are expected to be able to (1) determine the behavior of a C program that uses command-line arguments `argc` and `argv`; and (2) explain potential issues when using unsafe functions such as `strcpy()`.

The activity presents a C program that accepts command line parameters via `argc` and `argv` and displays them onto the console as a model through which students explore command line parameters in C. After some initial observations, students are asked to execute the program multiple times, with

different parameters each time as specified in the activity. After each execution, students make note of the number of parameters passed to the program, the value of `argc` and the number of elements in `argv`. They then answer a few questions through which they discover what the first element of `argv` contains and the correlation between `argc` and the number of strings passed to the program as parameters.

Next, the activity presents a second model, namely a program that creates a ten-byte character array and two strings, one with five characters and one with more than 15 characters. The program then performs a series of different `strcpy()` functions, copying the short string, the long string and a user-specified command line parameter to the ten-byte buffer. After asking students to make some observations about the buffer and strings, the activity asks students to execute the program with one of the `strcpy()` calls at a time and observe the program's behavior, including scenarios that result in a buffer overflow. By the end of the activity, students discover what a buffer overflow is and how it may be caused.

C. Process Memory Layout

Upon completing this activity, students should be able to (1) determine when and at what position function data is added to and removed from the stack corresponding to the execution of functions within a program; (2) describe the purpose, relative positions, growth directions and limits of multiple segments within a program's main memory; and (3) determine the layout and values of a function's stack frame based on its parameters, local variables and its invocation within a program.

The activity presents the model shown in Fig. 1. Students answer questions exploring the model and eventually discover when and at what positions stack frames are added to and removed from the stack, corresponding to the execution of functions with a program, in which directions the stack grows / shrinks as program execution proceeds and identify the state of the stack for any given point of execution within the program.

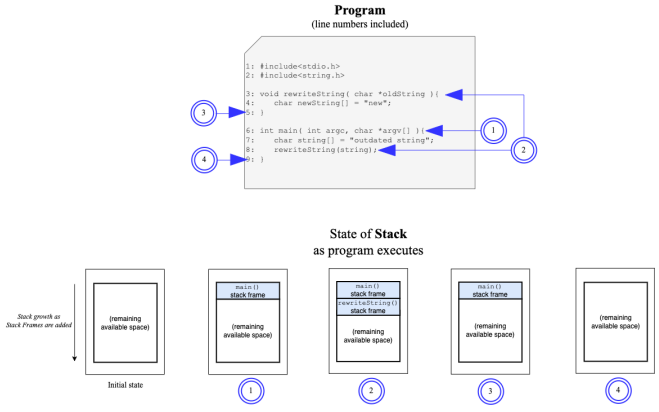

Fig. 1: Process Memory Layout Activity - Model 1

Next, the activity presents a second model that depicts a higher-level view of a computer's main memory. The main memory has the Stack, Heap, Data and Code regions marked, with the growth directions for the Stack and Heap indicated. The model also depicts the register file within the CPU, in which the Stack Pointer is explicitly indicated and points to the top of the Stack. By exploring this model and connecting things back to Model 1, students discover the purpose, relative positions, growth directions and limits of the Stack, Heap, Data and Code segments and the purpose of the Stack Pointer. They are also able to determine the state of the Stack and Stack Pointer as program execution proceeds.

The activity then presents a third model that highlights one specific program execution point, the corresponding state of the Stack and then expands the stack frame of the function currently on the top of the Stack. Students explore this model to understand the general layout of a stack frame and the purpose of each portion of the stack frame. Students are eventually able to determine to which program execution point the return address within a stack frame should point and the layout and values of a given function's stack frame based on its parameters, local variables and its invocation within a program.

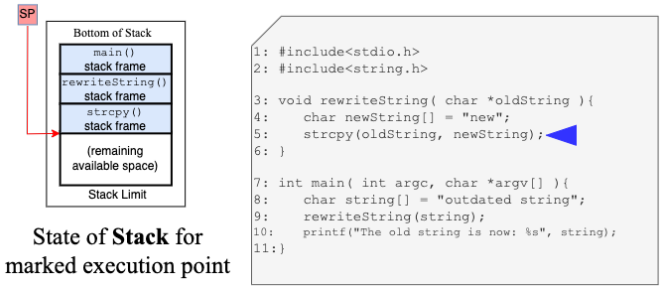

Fig. 2: Relative positions of memory

D. Stack Smashing

This activity aims to teach students how to complete a stack smashing attack using concepts learned in the previous two activities. Upon completing this activity, students are expected to (1) recognize unsafe user inputs that can overwrite useful stack frame contents and result in a stack smashing attack; (b) be able to calculate the payload size needed to overwrite the return address section of a given stack frame; and (3) be able to explain the purpose of the NOP sled and repeated return address in facilitating a stack smashing attack in practice.

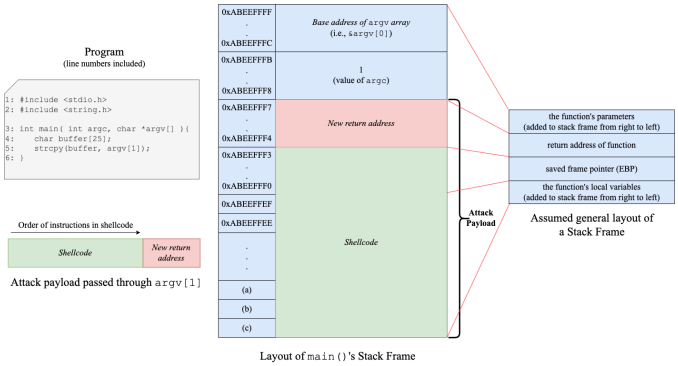

Fig. 3: Stack Smashing Activity - Model 2

The activity presents a model that depicts a very simple program that uses `strcpy()` to copy `argv[1])` to a local buffer, the current state of the stack just before `strcpy()` is invoked and the layout of the `main()` function at that instant. Students explore the model, recognizing the point of program execution and identifying the starting and ending memory addresses of each section of `main()`'s stack frame. Through further questions, they discover how the stack frame state will be updated after `strcpy()` executes and how long the string passed through `argv[1])` must be in order to fully overwrite the return address on `main()`'s stack frame.

Next, the activity formally defines what a Stack Smashing attack is and what shellcode is. The activity then presents the model shown in Fig. 3, depicting a basic stack smashing attack payload and its position on `main()`'s stack frame after `strcpy()` executes. Students calculate the starting memory address for the attack payload. Next, through a series of questions, they discover how long the attack payload would need to be to overflow buffers of different sizes (including the one shown in Fig. 3) and overwrite the return address and what the new return address must be in order to direct program execution to the start of the shellcode within the attack payload.

Finally, the activity presents a model that depicts a more sophisticated attack payload containing a NOP sled, shellcode and a repeated return address that aims to land somewhere within the NOP sled (see Fig. 4). Students explore the model and then discover the role of the NOP sled and repeated return address in maximizing the changes of a successful stack-smashing attack. Through further questions, they apply what they have learned so far to identify what types of payloads (i.e., what lengths of NOP sleds and how many repeats of the return address) are most likely to result in a successful attack.

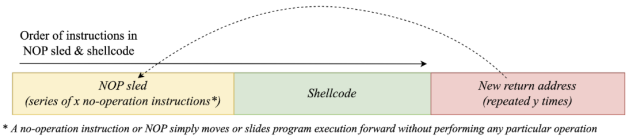

Fig. 4: Stack Smashing Attack Payload

E. Stack Smashing Defenses

At the end of this activity, students are expected to be able to (1) recognize the main techniques used to protect against stack-smashing attacks, namely *Address Space Layout Randomization* (ASLR) [12], *non-executable stacks* [14], *stack canaries* [16], and safer C string functions, (2) explain how and why ASLR, non-executable stack and stack canary can help prevent stack smashing attack; and (3) explain why `strncpy()` is safer than `strcpy()`.

The first model in this activity presents two systems and an attacker's deduction of the approximate start address of a vulnerable buffer (e.g., binary analysis) over multiple execution attempts for the two systems. Through a series of questions, students explore the model and observe that the approximate address of the buffer across multiple executions of the first system are very close to each other in memory. On the other hand, students observe that the other system has drastically different address approximations after each execution attempt. From these observations, they infer that a stack smashing attack is much less likely to succeed in the second system when compared to the first. They are then formally introduced to the concept of ASLR and informed that the second system uses this approach. After this, students answer a series of questions through which they solidify their understanding of how ASLR works and which step of the stack smashing process it targets.

The activity then presents a second model that depicts the placement of a copy of some reference value on the stack frame of a function. Students explore this model and infer under what circumstances the copy of the reference value may change during function execution and how that could be used to detect when a stack smashing attack is being attempted. They are then formally introduced to the concept of a stack canary, a known value that the system places between a vulnerable buffer and other important stack frame elements such as the return address, which tells the system whether those values have been overwritten. The activity concludes with a few questions and information through which students discover how non-executable stacks can prevent stack smashing attacks and why `strncpy()` is safer than `strcpy()`.

F. Active Learning Exercise with Visualization Tool

DISSAV is a program visualization tool designed to help students visualize the process of a stack smashing attack. DISSAV guides students through constructing a stack smashing attack in three phases, "Create the Program", "Construct the Payload", and "Execute the Program". DISSAV allows students to customize vulnerable functions and choose from a list of dummy attacker actions, such as "Start a remote shell" or "Wipe OS", through dynamic input.

We design an active-learning exercise to complement DISSAV. The activity starts with fundamental C programming concepts and then guides students in creating a vulnerable function, constructing a payload, and executing it, all within DISSAV. Emphasizing experimentation, students test various inputs to understand how computers handle data on the stack. While the activity offers instructions and hints for stack smashing attack payload construction, students must experiment with factors like the number of NOP sleds and identifying the correct return address. This trial-and-error approach mirrors real stack smashing attacks. The exercise includes questions covering key variables on the call stack to underscore their significance. Finally, high-level questions prompt students to reflect on their process, reinforcing essential concepts in stack smashing attacks. Ultimately, the exercise aims to lead students to comprehend abstract concepts like data passing and shellcode execution on the call stack effectively.

IV. Study Design

To evaluate student perceptions of our software security module and answer the research questions outlined in Section I, we conducted a user study approved by our Institutional Review Board (IRB). We integrated the module into an undergraduate cybersecurity course and administered a voluntary survey in order to gauge students' attitudes and perceptions. The course encompasses a broad range of security topics and is required for a significant portion of our program's students, who must have completed a prerequisite course in data structures and algorithms. Preceding introductory programming and data structures courses use Java, so many students entering this cybersecurity course may lack exposure to C programming.

1) Data Collection: The source for data collection is a voluntary student survey that consists of 31 *attitudinal* questions with Likert-scale responses ranging from Strongly Agree to Strongly Disagree, with 17 questions about the guided learning activities and 14, about DISSAV and the accompanying active-learning exercise[1]. These seek to elicit student opinions on various aspects of the activities and tool. We ask questions about guided learning activity structure and outcomes, engagement, length and challenge level, and team role usage. For DISSAV and the exercise, we ask questions about the user interface, student learning through them, engagement, and length and challenge. This structured format allows students to express the intensity of their attitudes and perceptions, enabling comprehensive insights into and a systematic evaluation of student satisfaction and areas for improvement.

2) Deployment: We deployed our software security module and survey across multiple sections of an undergraduate cybersecurity course during Fall 2022, Spring 2023, and Fall 2023. Table I shows the number of students enrolled each semester and the number who consented to participate in our study.

TABLE I: Student Enrollment and Participation

Semester	# of Students Enrolled	# of Consenting Students
Fall 2022	89	77
Spring 2023	32	25
Fall 2023	99	27

3) Data Analysis: We use descriptive statistics to analyze our Likert-scale survey responses, presenting the distribution, median, and mode of student responses.

V. Results

In this section, we present and discuss the distribution of student responses to our Likert-scale questions using descriptive statistics, one category of questions at a time. For each question, the distribution of student responses ranges from Strongly Agree (5) to Strongly Disagree (1). In all our response distribution graphs, the y-axis shows the percentage for each type of response. Along the x-axis, we present a bar for each survey question and semester. Each bar is ordered such that the 'Strongly Agree' response appears at the top and 'Strongly Disagree', at the bottom. A black border represents the *median* (i.e., middle) response within each bar. In most cases, the *mode* (i.e., most frequent) response is also the same as the median. In cases where the two values are different, the mode is shown with a red border.

A. Guided Learning: Activity Structure & Outcomes

We ask students for their perceptions on the clarity / structure of the models and questions within our four guided learning activities, whether students felt that the activities helped them learn the targeted concepts, and whether they eventually felt that they achieved the intended learning outcomes. Fig. 5 shows the distribution of student responses for this category of questions. We see that a majority of students across all semesters have positive perceptions about the visual clarity and consistency of the models (ASO-2), feel that the activity questions draws their attention to key information in the models (ASO-7), and have a positive perception about the clarity of model content (ASO-1) and the progression of questions (ASO-6). These are very encouraging, because they directly align with key goals of POGIL-style activities. Student perception of the phrasing of activity questions (ASO-5) was low in Fall 2022 (31%), but gradually increased over the semesters, rising to 68% in Fall 2023. While the increase in positive perceptions across semesters is encouraging, the generally lower percentages suggest that improvements may need to be made in the phrasing of questions. ASO-3, ASO-4, and ASO-8, which are all related to perceived learning of targeted concepts through the guided learning activities, have lower positive perceptions in Fall 2022 and Spring 2023, but show improvement in Fall 2023. The guided learning style was new to the students and the course instructional team in this study. Students may have come in with an expectation that instructors will provide lecture material (live lecture, pre-class video, etc.) rather than teach entirely via activities, which may have influenced their perceptions. Over the semesters, the course instructional team has been attempting to set the context for and facilitate the activities better, which may have improved perceptions over the semesters.

B. Guided Learning: Activity Length & Challenge Level

We ask whether students felt that the length and challenge level of the activities were appropriate. Fig. 6 shows the distribution of student responses for this category of questions. In Fall 2022, we see that almost 40% of students feel that the activities were too challenging (ALC-1), but this percentage decreases over the next two semesters. Once again, this is likely due to the instructional team becoming more familiar with facilitating POGIL-style activities with time. On the other hand, a very small percentage of students feel that the activities were not challenging enough (ALC-2).

We see that approximately a third of the students across all semesters feel that the activities were too long for the time given to complete them and a little less than a third are neutral (ALC-3), but that less than 10% feel that they are too short

[1]Our survey also contains *open-ended* and *demographic* questions. In this paper, we focus only on the Likert-scale questions.

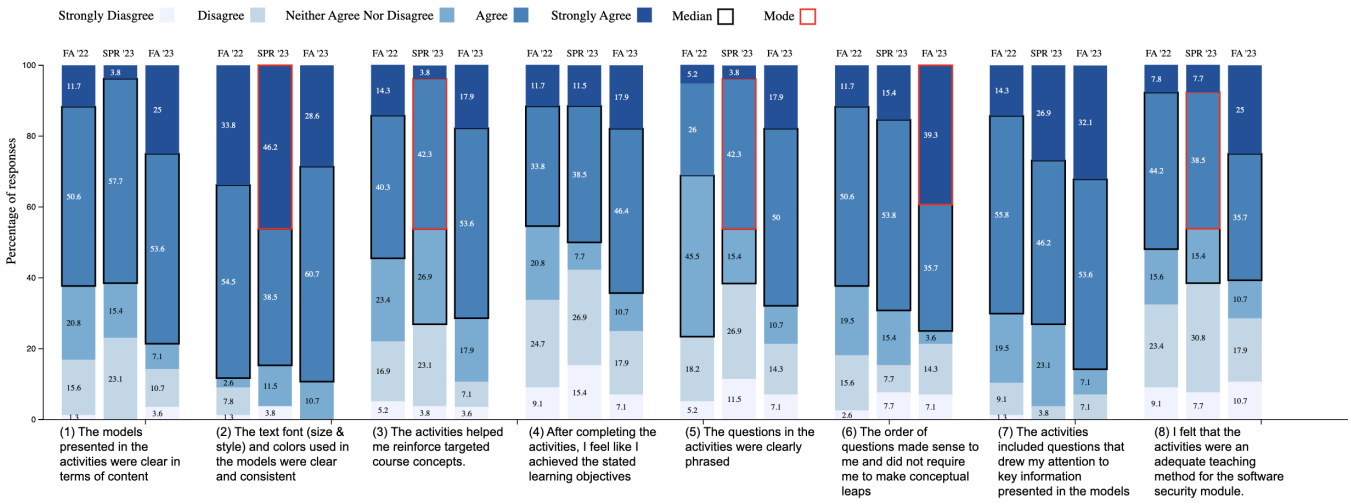

Fig. 5: Student response distribution for Activity Structure & Outcomes (ASO) questions

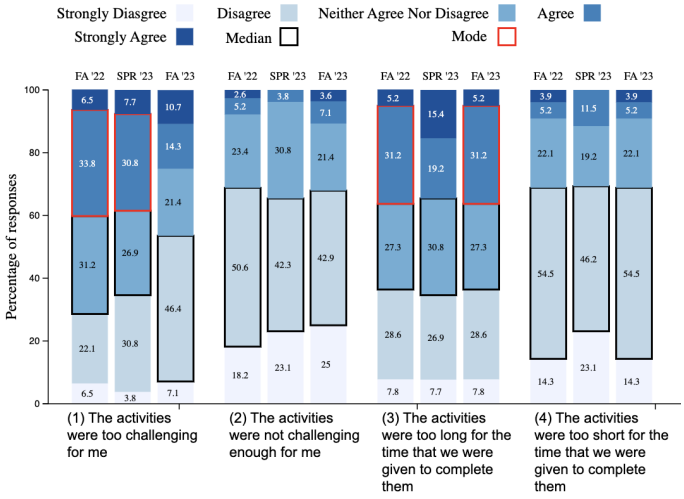

Fig. 6: Student response distribution for Activity Length and Challenge Level (ALC) questions

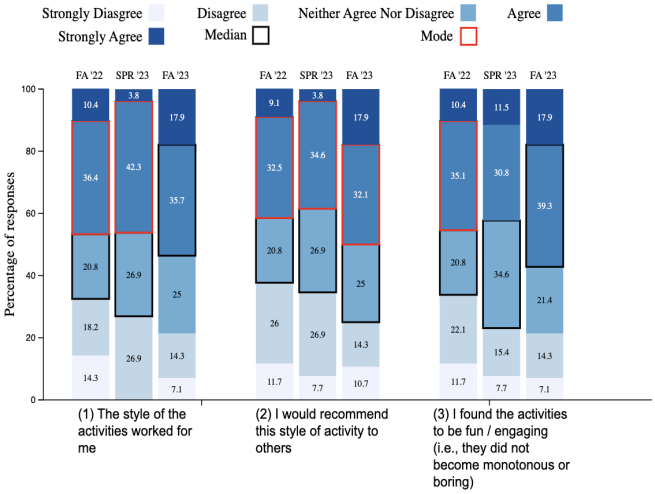

Fig. 7: Student response distribution for Activity Style & Engagement (ASE) questions

(ALC-4), with the median being neutral and mode being on the 'Agree' side. We designed our activities with the intention of having them be completed within the class time of 75 minutes. However, this student feedback suggests a need for us to either shorten / break up the activities, or provide additional time for them. In order to address the issue on an immediate basis during the study duration, the instructional team allowed students additional time after the class to complete and submit the activities. While this is not ideal because it does not allow for teamwork, we took this approach to avoid grade reductions due to time constraints.

C. Guided Learning: Activity Style & Engagement

Fig. 7 shows the distribution of responses for questions about how engaging students found the guided learning activity style to be. Across all three semesters, the vast majority of students have neutral to positive perceptions of the activity style, with around 40-50% agreeing that the activity style worked for them and was engaging. However, there is a significant proportion of students (around 30-40%) who are not engaged or satisfied with the activity style, suggesting that students may not be particularly excited about the guided learning style of activities. In the future, we will introduce the purpose and goals of this style more explicitly, to set up clearer expectations and highlight the advantages of the style.

D. Guided Learning: Team Role Usage

We ask students whether they used team member roles as suggested and whether it helped. Fig. 8 shows the distribution of student responses. We see that in Fall 2022, only around

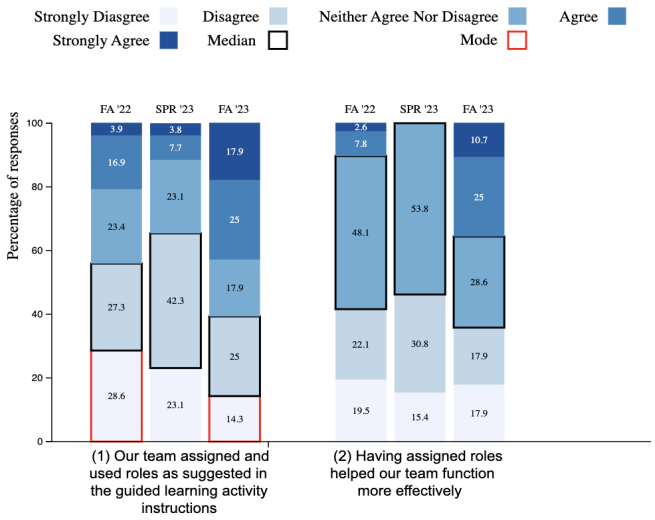

Fig. 8: Response distribution for Team Role Usage (TRU)

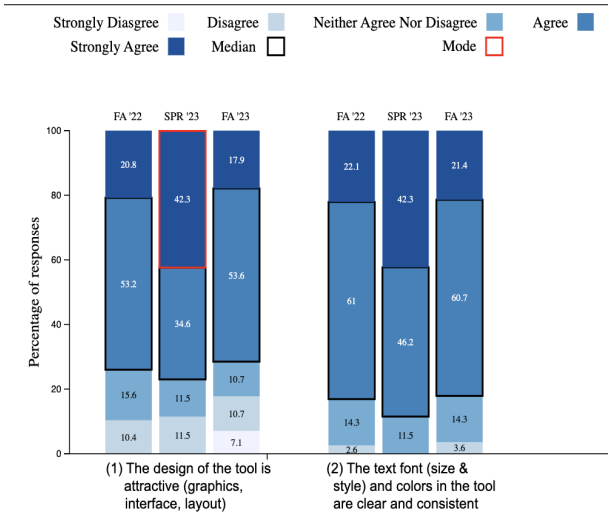

Fig. 9: Response distribution for User Interface (UI)

20% of students use team roles (TRU-1), which explains the high level of neutral and negative responses to TRU-2. Less than 10% of students feel that team roles help them. The proportion of the positive responses for TRU-1 increased in Fall 2023, with roughly 40% suggesting that more teams used assigned roles. However, neutral and negative responses to TRU-2 in Fall 2023 add up to more than 50%, indicating that roughly half the teams who used team roles did not find it helpful. In the future, we will model the use of team roles, require their use, and ask students to record team roles as part of their submission. We will also enhance the activities to incorporate team roles more explicitly into activity prompts.

E. DISSAV: User Interface

From Fig. 9, we see that over 75% of students have positive perceptions about the user interface of DISSAV, which suggests that the tool is well-designed and accessible.

F. DISSAV: Learning Experience

We ask for students' perception of their learning of targeted concepts through DISSAV and the accompanying exercise. Fig. 10 shows the distribution of student responses. Student perceptions are consistently positive for this category, with 60 - over 80% of students conveying positive perceptions.

G. DISSAV: Activity Engagement and Motivation

We ask students about their level of engagement / enjoyment of DISSAV and its accompanying exercise. A vast majority of students (over 75%) find the activity to be fun and over 60% feel a sense of satisfaction upon completing the phases of the exercise. On the other hand, the tool and exercise were not so engaging that students lost track of time.

H. DISSAV: Activity Length and Challenge Level

For activity length and challenge level, from Fig. 12, we see that students feel that the exercise was appropriately challenging and neither too long nor too short.

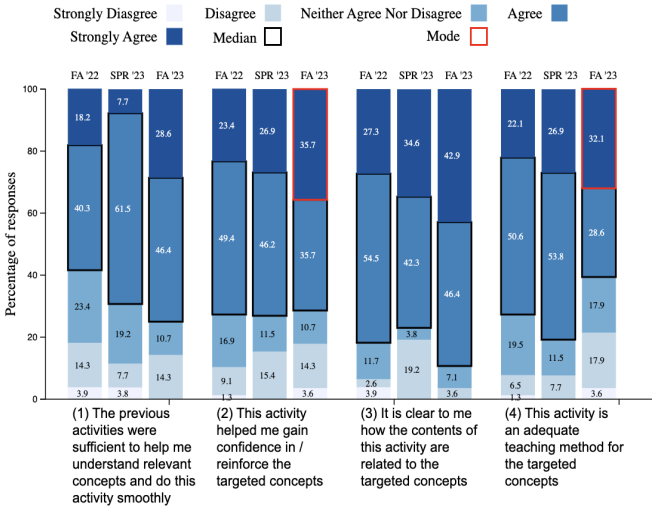

Fig. 10: Response distribution for Learning Experience (LE)

I. DISSAV: Activity Style and Recommendation

From Fig. 13, we see that a majority of students have positive perceptions towards the style and would recommend it to others, which is very encouraging.

VI. Discussion

a) Analysis summary: Student responses to our attitudinal survey suggest that they mostly find the guided learning activities, DISSAV and the accompanying exercise useful, well-designed, accessible, and engaging, thus answering our two research questions in a positive manner. However, improvements are needed in terms of explaining the purpose and advantage of POGIL-style activities, activity question phrasing, teamwork facilitation, and activity length / time provided. Our survey contains free-response and demographic questions as well. In future work, we plan to conduct a *statistical analysis* to

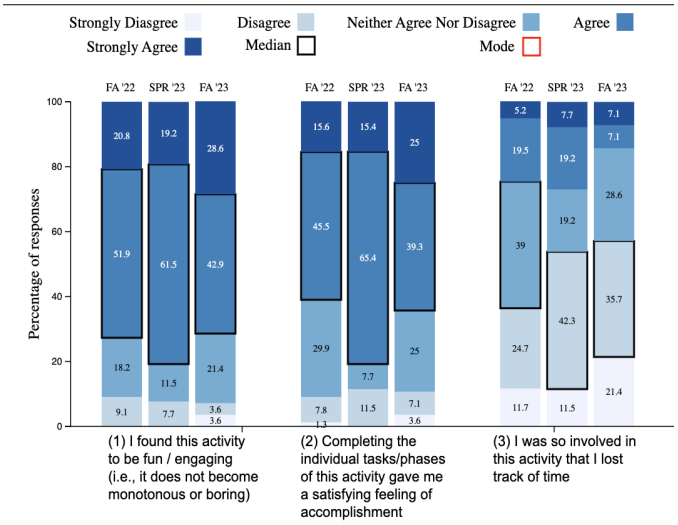

Fig. 11: Response distribution for Activity Engagement and Motivation (AEM)

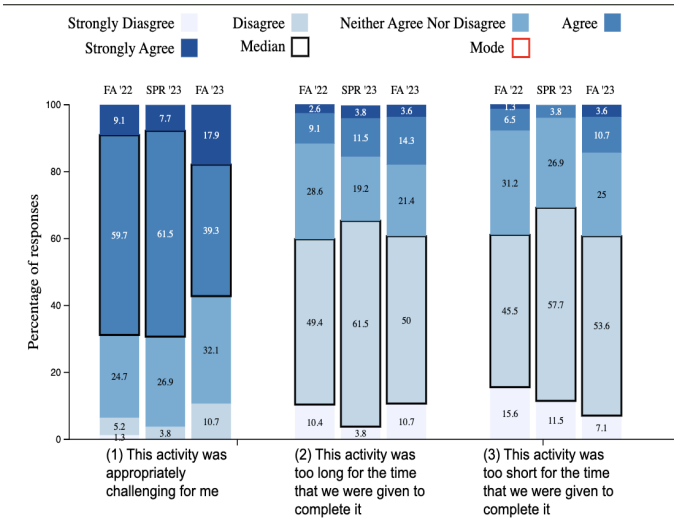

Fig. 12: Response distribution for Activity Length and Challenge Level (ALC-PV)

identify whether there are significant differences in student perceptions of our module across different demographics, including age, gender, and prior experience in C programming, stack smashing, program visualization tools. We also plan to conduct a *thematic analysis* of student responses to open-ended questions, to gain deeper insights into the strengths and areas of improvement for our module. We will also consider conducting in-depth focus groups or interviews with students.

b) Limitations: A limitation of our study is that we do not compare our outcomes with a control group. Although there were two sections of the class being taught by the same instructor in Fall 2022 and Fall 2023, we intentionally decided *not* to use one of the sections as a control group for our study. Prior research has found that using POGIL-style activities

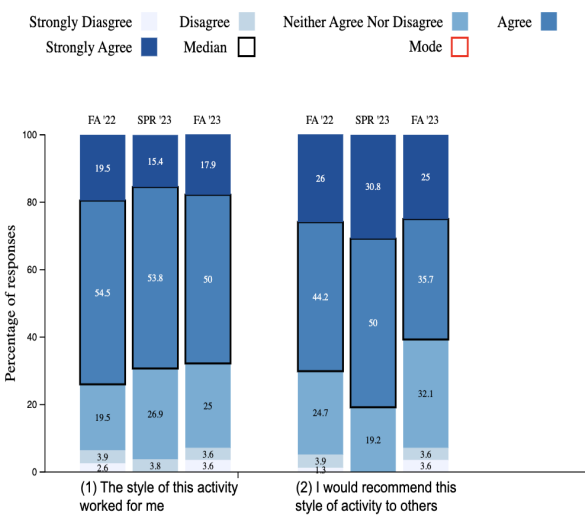

Fig. 13: Response distribution for Activity Style and Recommendation (ASR)

can improve students' sense of belonging, teamwork, and problem-solving skills, and improve their overall performance and mastery of content [7], [11], [17], [26]. So, we believe it would be unethical of us to knowingly deprive students within the control group of access to this style of activities. In future work, we will explore alternative research designs to determine if there is one that is both ethically viable and also provides us with comparison data. We also plan to compare student performance on course assessments in the Fall 2022, Spring 2023, and Fall 2023 semesters (i.e., from our current study) with that on similar assessments in *prior* semesters, before our guided learning style activities were created and deployed.

VII. Conclusion

In this paper, we briefly present our software security module comprising a suite of four guided learning activities, and a program visualization tool with an accompanying active-learning exercise, to teach stack smashing attacks and defenses. In addition, we present the results of a user study deploying this module in multiple sections of an undergraduate, introductory cybersecurity course in Fall 2022, Spring 2023, and Fall 2023. Our study finds that students have mostly positive perceptions about activity structure / design, content, and level of engagement, but that improvements may be needed to some aspects, including setting the context for the activities, question phrasing, activity length, and teamwork facilitation. In future work, we plan to conduct a statistical analysis of student demographic data and a thematic analysis of student responses to open-ended survey questions.

Acknowledgements

This research was supported in part by NSF-DGE #1947295 and UNC Charlotte.

REFERENCES

[1] Erik Akeyson, Harini Ramaprasad, and Meera Sridhar. Dissav: A dynamic, interactive stack-smashing attack visualization tool. In *Journal of The Colloquium for Information Systems Security Education*, volume 9, pages 8–8, 2022.

[2] Hanan Alshaher, Xiaohong Yuan, and Sajad Khorsandroo. Teaching flooding attack to the sdn data plane with pogil. In *Proceedings of the 21st Annual Conference on Information Technology Education*, pages 194–199, 2020.

[3] Dan Budny, Laura Lund, Jeff Vipperman, and JLIII Patzer. Four steps to teaching C programming. In *Proceedings of the 32nd IEEE Annual Frontiers in Education Conference*, volume 2, 2002.

[4] Luca Caviglione, Steffen Wendzel, Aleksandra Mileva, and Simon Vrhovec. Guest editorial: Multidisciplinary solutions to modern cybersecurity challenges. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 12(4):1–3, 12 2021.

[5] CS POGIL Project Team. Process oriented guided inquiry learning in computer science.

[6] Susan Gerhart, Jan Hogle, and Jedidiah Crandall. Demonstrating security vulnerabilities: the buffer overflow problem. In *Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, pages 980–983, 2003.

[7] Olga Glebova, Kendra Walther, and Clif Kussmaul. Guiding students to discover cs concepts and develop process skills using pogil. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*, pages 1198–1198, 2022.

[8] Wu He, Li Xu, Yuming He, Xiaohong Yuan, Li Yang, and Jennifer Ellis. Teaching buffer overflow via a guided inquiry collaborative learning activity. In *SAIS Proceedings*, 2020.

[9] Yuming He, Wu He, Lida Xu, Xin Tian, Xiaohong Yuan, Li Yang, and Jennifer T Ellis. Guided inquiry collaborative learning (GICL) for online teaching in cybersecurity: Challenges and recommendations. In *Journal of The Colloquium for Information Systems Security Education*, volume 9, pages 5–5, 2022.

[10] Egan Heinsen and Chris McDonald. Program visualization and explanation for novice C programmers. *Proceedings of the Sixteenth Australasian Computing Education Conference*, 148:51–57, 2014.

[11] Helen H Hu, Aman Yadav, Donna M Gavin, Clif Kussmaul, and Chris Mayfield. Teamwork in cs1: Student learning and experience with pogil. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 729–735, 2023.

[12] Yeongjin Jang, Sangho Lee, and Taesoo Kim. Breaking kernel address space layout randomization with intel TSX. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 380–392, 2016.

[13] Nadia Jones, Qingrui Yu, Karen Schell, and Huiming Yu. Teaching secure program design. In *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering*, pages 3–5, 2019.

[14] G.S. Kc and A.D. Keromytis. e-NeXSh: achieving an effectively non-executable stack and heap via system-call policing. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC'05)*, pages 15 pp.–302, 2005.

[15] David R. Krathwohl. A revision of bloom's taxonomy: An overview. *Theory Into Practice*, 41(4):212–218, 2002.

[16] Hans Liljestrand, Zaheer Gauhar, Thomas Nyman, Jan-Erik Ekberg, and N. Asokan. Protecting the stack with paced canaries. In *Proceedings of the 4th Workshop on System Software for Trusted Execution*, SysTEX '19, 2019.

[17] Sukanya Kannan Moudgalya, Chris Mayfield, Aman Yadav, Helen H Hu, and Clif Kussmaul. Measuring students' sense of belonging in introductory cs courses. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 445–451, 2021.

[18] Aleph One. Smashing the stack for fun and profit. *Phrack magazine*, 7(49):14–16, 1996.

[19] Chang Phuong, Noman Saied, and Li Yang. A hands-on education framework for cybersecurity. In *Proceedings of the IEEE Frontiers in Education Conference*, pages 1–5, 2023.

[20] POGIL Project Team. Effectiveness of process-oriented guided inquiry learning.

[21] POGIL Project Team. Process-oriented guided inquiry learning.

[22] Danijel Radošević, Tihomir Orehovački, and Alen Lovrenčić. New approaches and tools in teaching programming. In *Proceedings of the Central European Conference on Information and Intelligent Systems (CECIIS)*, 2009.

[23] Dino Schweitzer and Jeff Boleng. A simple machine simulator for teaching stack frames. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, pages 361–365, 2010.

[24] Xin Tian and Zhigang Li. Collaborative learning for information security topics: A pilot study. In *AMCIS*, 2020.

[25] James Walker, Man Wang, Steve Carr, Jean Mayo, and Ching-Kuang Shene. A system for visualizing the process address space in the context of teaching secure coding in c. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 1033–1039, 2020.

[26] Aman Yadav, Chris Mayfield, Sukanya Kannan Moudgalya, Clif Kussmaul, and Helen H Hu. Collaborative learning, self-efficacy, and student performance in cs1 pogil. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 775–781, 2021.

[27] Jeong Yang and Akhtar Lodgher. Fundamental defensive programming practicec with secure coding modules. *International Conference on Security and Management*.

[28] Li Yang, Xiaohong Yuan, Wu He, Jennifer Ellis, and Jonathan Land. Cybersecurity education with pogil: Experiences with access control instruction. *Journal of The Colloquium for Information Systems Security Education*, 6(2), 2019.

[29] Xiaohong Yuan, Tian Zhang, Ali Albu Shama, Jinsheng Xu, Li Yang, Jennifer Ellis, Wu He, and Cynthia Waters. Teaching cybersecurity using guided inquiry collaborative learning. In *Proceedings of the IEEE Frontiers in Education Conference*, pages 1–6, 2019.

[30] Jinghua Zhang, Xiaohong Yuan, Jaris Johnson, Jinsheng Xu, and Mounika Vanamala. Developing and assessing a web-based interactive visualization tool to teach buffer overflow concepts. In *Proceedings of the IEEE Frontiers in Education Conference*, pages 1–7, 2020.