Interactive Program Visualization to Teach Stack Smashing: An Experience Report

Harini Ramaprasad
Computer Science
Univ. of North Carolina at Charlotte
Charlotte, NC, USA
hramapra@uncc.edu
0000-0002-1598-4677

Meera Sridhar
Software and Information Systems Univ. of North Carolina at Charlotte
Charlotte, NC, USA
msridhar@uncc.edu
0000-0002-7508-5024

Erik Akeyson
Software and Information Systems Univ. of North Carolina at Charlotte
Charlotte, NC, USA
eakeyson@uncc.edu

***Abstract*—This paper presents an experience report on using an interactive program visualization tool — Dynamic, Interactive Stack-Smashing Attack Visualization (DISSAV) — and a complementary active-learning exercise to teach stack smashing, a key software security attack. The visualization tool and active-learning exercise work synergistically to guide the student through challenging, abstract concepts in the advanced cybersecurity area. DISSAV and the exercise are deployed within the software security module of an undergraduate cybersecurity course that introduces a broad range of security topics.**

A study is designed that collects and evaluates student perceptions on the user interface of DISSAV and the effectiveness of the two resources in improving student learning and engagement. The study finds that over 80% of responses to user interface questions, 66% of responses to student learning questions and 64% of responses to student engagement questions are positive, suggesting that the resources improve student learning and engagement in general. The study does not find discernible patterns of difference in responses from students of different ages and varying levels of prior experience with stack smashing attacks, program visualization tools and C programming.

Keywords—program visualization, stack smashing attacks, active-learning, engaged pedagogy, user study

I. INTRODUCTION

Stack smashing attacks [1] are a dangerous class of software attacks that an attacker can use to hijack the control-flow of a program [2], [3] to execute arbitrary code on the victim machine. Although stack smashing attacks only affect languages with unsafe functions, they have widespread impact due to the large amount of legacy code used in today's applications [4] – [7] and are still quite widespread [8], [9]. Factors such as *patch lag* [10] — the time it takes for a user to update to the most recent software version, and device *end-of-life* [11] — when vendors stop maintaining the firmware or software for a device, including providing bug fixes and security patches — increase the prevalence of stack smashing vulnerabilities in devices that use legacy code.

Stack smashing attacks are a critical class of attacks to teach cybersecurity students today, due to their large impact and potential danger [12]. However, teaching stack smashing can be an arduous task for cybersecurity educators due to the immense background required (including process memory layout, call stacks, buffer storage and overflows, memory address arithmetic, shellcode), the steep learning curve that the C language imposes on new programmers, and the painstaking memory address space calculations involved [1], [13] – [16].

In prior work, we presented a program visualization tool — Dynamic, Interactive Stack-Smashing Attack Visualization (DISSAV) — that aims to address the above challenges and teach students about stack smashing attacks through a guided, simulated attack scenario [17]. In this paper, we report on the *evaluation* of DISSAV and an accompanying, complementary active-learning exercise.

Through our evaluation, we aim to answer three research questions: **(R1)** Do students find that DISSAV and the active-learning exercise improve their learning of stack smashing? **(R2)** Do students find DISSAV and the active-learning exercise to be engaging resources for learning about stack smashing? **(R3)** Do DISSAV and the active-learning exercise consistently improve students' perceived learning and engagement across all age groups and genders, including students with no prior experience on the topic?

To answer the above questions, we deployed DISSAV and the active-learning exercise in two sections of a junior level undergraduate course at UNC Charlotte named *Introduction to Security and Privacy* in Fall 2021, with a total of 104 students. We also designed and administered a student survey to obtain student feedback on DISSAV's interface and on their perceived learning and engagement with the tool and accompanying exercise.

The student survey consists of two user interface questions, six student learning questions, and six student engagement questions. On an average, we find that over 80% of responses to User Interface questions, over 66% of responses to Student Learning questions and over 63% of responses to Student Engagement questions are positive while 0%, under 4% and under 10% of the responses within the same categories are negative. This indicates that, while there are aspects that need improvement, DISSAV and the active-learning exercise are generally beneficial and engaging resources to learn about stack smashing.

The main contributions of this paper are:

- We deploy DISSAV and an accompanying active-learning exercise in the secure software module of two sections of an introductory cybersecurity course in Fall 2021, with 104 students.
- We formulate three research questions to evaluate the effectiveness of DISSAV and the exercise in improving student learning and engagement across multiple demographic groups.
- We design and deploy a survey with Likert-scale, open-ended and demographic questions to answer our research questions.
- We evaluate the effectiveness of DISSAV and the exercise through a systematic analysis of survey data.

II. BRIEF BACKGROUND OF EVALUATED RESOURCES

DISSAV [17] 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 is interactive and engaging, making use of colors, fonts, icons, buttons and more to improve student engagement, and appeal to a broader and more diverse student audience. DISSAV offers students the ability to customize an attack scenario (within limits), and provides guided, incremental steps for completing the attack. A main highlight of DISSAV is that it provides dynamic visualization, displaying the current state of the call stack during program execution, including a drop-down button to visualize details about the current stack frame. DISSAV also highlights various parts of the program code itself during execution. DISSAV helps visualize memory addresses and contents of the stack frames, an abstract concept for students. DISSAV also allows students to customize vulnerable functions and choose from a list of final attacker actions, such as “Start a remote shell” or “Wipe OS”, through dynamic input.

We design and deploy an active-learning exercise to accompany DISSAV. The exercise starts by covering simple C programming concepts (e.g., data types) then continues to the three phases mentioned above. The activity provides instructions on creating a vulnerable function, constructing a payload, and executing the function. The activity encourages students to use “different strings of different lengths and number of words” before attempting to construct an attack payload. We incorporate this feature to allow students to test different inputs and to experiment and visualize how the computer passes and stores data on the stack before constructing a full payload. While the activity provides instructions for payload construction along with hints, the exact process is not given. Students must experiment by using different numbers of *NOP sleds* (“no-operation instructions”) [1], identifying and placing the correct malicious return address (location in memory to jump to, to execute the malicious code), and formatting that return address. We encourage students to learn how the return address is overwritten and how the shellcode is executed through trial and error, similar to a real stack smashing attack. We include questions that cover major variables on the call stack (e.g., `argv` and the character ‘\x’) to highlight their importance. At the end of the activity, we provide more high-level questions — e.g., how did they determine their new return address, how did they determine the length of the payload, etc. — with the aim of emphasizing key concepts in a stack smashing attack. The aim of the exercise is to have students build up to these more abstract concepts such as how the computer passes data from `argv` to `main`’s stack frame and the execution of shellcode on the call stack to adequately understand stack smashing attacks.

III. STUDY DESIGN AND DEPLOYMENT

To evaluate the effectiveness of DISSAV and the accompanying active-learning exercise in improving student learning and engagement, we designed a user study approved by our Institutional Review Board or IRB. We deployed the two resources in an undergraduate cybersecurity course and used a voluntary survey to gather student perceptions on them. Through this study, we aim to answer the research questions listed in Section I. In addition, we aim to get feedback on DISSAV’s usability (user interface, ease of use, etc.).

A. Student Survey

We design our student survey to include fourteen 5-point Likert scale questions, with answer options ranging from Strongly Agree (Weight: 5), to Strongly Disagree (Weight: 1), two free-response questions, and some demographics questions, as described below.

- The Likert scale questions ask for student feedback on DISSAV’s user interface (two questions) and on student learning (six questions) and engagement (six questions) with DISSAV and the accompanying exercise. These questions allow us to quantitatively evaluate student opinions and attitudes towards the tool and exercise.
- The two free response questions ask students to list two strengths and two weaknesses of the tool and exercise, respectively. We intentionally ask for a specific number of strengths / weakness to encourage more concrete responses from students as opposed to having a fully open-ended question about their experiences.
- The demographic questions ask students to select an age range, gender, and indicate their experience level in three different areas, namely C programming, use of program visualization tools, and stack smashing.

B. Deployment

We deployed DISSAV, the accompanying active-learning exercise and the voluntary student survey within the software security module of two sections of a junior level undergraduate course named *Introduction to Security and Privacy* at UNC Charlotte in Fall 2021. The course introduces a broad range of security topics and is a required course for a large number of students in our program. The course has a *Data Structures and Algorithms* prerequisite. The prerequisite course and prior introductory programming courses at UNC Charlotte are currently taught in Java, so students coming into the introductory cybersecurity course may not have taken any course that teaches or uses C programming. Among the 104 students who were enrolled in the two sections in Fall 2021, 26 students completed the voluntary survey and consented to have their responses collected and analyzed.

IV. RESULTS

In this section, we present our analysis of responses to the student survey from the 26 students who consented[1]. Table I shows our Likert scale questions and the distribution of responses for each question (percentages are rounded up / down to the nearest integer for readability). We classify the questions into three categories, namely User Interface, Student Learning and Student Engagement questions. In our analysis and graphs, we use the term *positive* to refer to responses of Strongly Agree or Agree, *neutral* for Neither Agree Nor Disagree, and *negative* for Disagree or Strongly Disagree.

For each category, we also discuss whether responses are different for students in different age groups and with / without prior knowledge in three relevant areas. Due to our small overall sample size (26) and some very small demographic groups, we do not perform statistical analysis of demographic data, but instead discuss the distribution of responses in general terms. We do not report or consider gender related data because we had an extremely small percentage of female participants, with the rest being male participants.

Before discussing each category of questions, we make general observations about our student demographics using figures that show the distribution of responses across demographic groups and which will be discussed in more detail later. Fig. 1a (and 2a) shows that over 73% of the students fall into the 18-22 age group[2], which is the most common age group for undergraduate students in general. Fig. 1b (and 2b) shows that over 80% of students have little to some prior C programming experience. Fig. 1c (and 2c) shows that more than 57% of our students do not have prior stack smashing knowledge, which is expected because our course is an introductory one in security and privacy. Finally, from Fig. 1d (and 2d), we observe that over 76% of our students have little to some experience using program visualization tools (in other contexts).

A. *User Interface*

From the User Interface section of Table I, we observe that more than 80% of the students have positive perceptions of the user interface of DISSAV and the rest are neutral, suggesting that DISSAV's user interface is mostly clear, consistent and attractive. For user interface related questions, we did not observe discernible differences between students with and without prior experience in the three areas or students from different age groups. So, we do not present those results.

B. *Student Learning*

To analyze perceived student learning, we look at data in the Student Learning section of Table I and Fig. 1a, 1b, 1c and 1d, which show the distribution of responses to the Student Learning questions for different age groups and varying levels of knowledge / experience with C programming, stack smashing and program visualization tools, respectively.

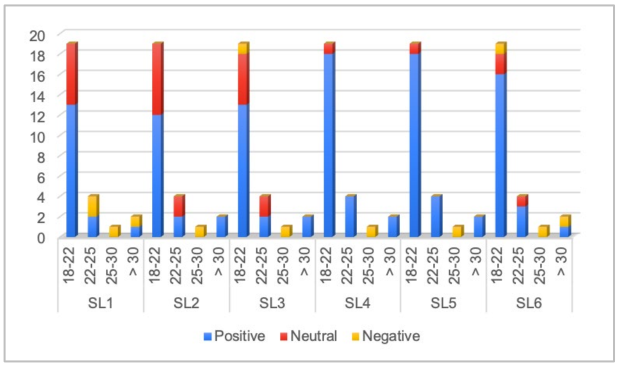
1) SL4 & SL5: From Table I, we observe that an overwhelming majority of students (24 out of 26 students, i.e., 96%) indicated that they found the contents of the activity relevant (SL4) and helpful (SL5) to learn the targeted concepts, with only one student being neutral and one being negative. From the SL4 and SL5 bars in Fig. 1a, 1b, 1c and 1d, we observe that the neutral response is from a student in the 18-22 age group with some prior C programming experience and a little knowledge / experience in stack smashing and program visualization tools and the negative response is from a student in the 25-30 age group with no C programming background and a little knowledge / experience in stack smashing and program visualization tools. The majority of responses, coming from students in varying age groups and with varying prior experience, are positive. This indicates that the activity is consistently relevant and helpful in learning targeted concepts.

2) SL1: Although still a majority, Table I shows that less students (65%) indicated that they found the learning content sufficient to complete the activity (SL1), with 23% being neutral and 12% being negative. The SL1 bars in Fig. 1a, 1b, 1c and 1d show that the neutral responses are from students in the 18-22 age group with a little or some C programming experience and with varying levels of stack smashing knowledge and program visualization tool experience. The negative responses come from students in older age groups with varying levels of C programming experience, stack smashing knowledge and none to little program visualization tool experience. The varying demographics suggest that there may generally be a need to provide more learning resources for background concepts before exposing students to DISSAV and the active-learning exercise where they put everything together and attempt a dummy stack smashing attack.

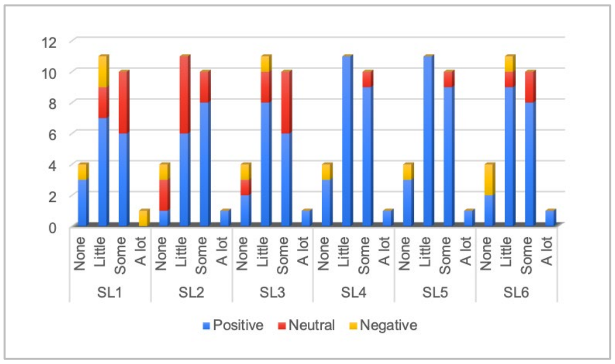
1. We did not conduct any analysis until after consent was verified and all our data was fully de-identified.
2. We acknowledge that some age groups have overlaps with the next. We will address this in future surveys.

TABLE I. STUDENT SURVEY QUESTIONS AND RESPONSES (N=26)

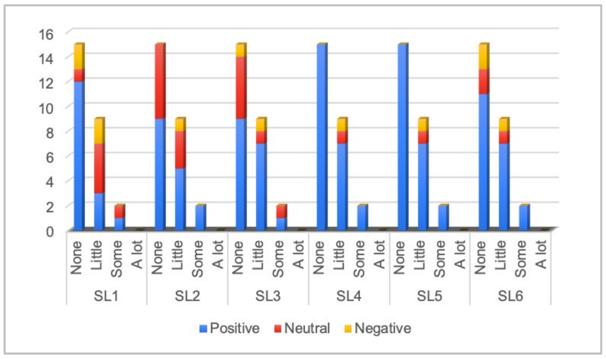
Question	**Strongly Agree**	**Agree**	**Neither A / D**	**Disagree**	**Strongly Disagree**
User Interface					
UI1: The application design is attractive (graphics, interface, layout)	7 (27%)	14 (54%)	5 (19%)	0 (0%)	0 (0%)
UI2: The text font (size and style) and colors are clear and consistent.	8 (31%)	15 (58%)	3 (11%)	0 (0%)	0 (0%)
Student learning					
SL1: The learning content and/or previous activities were sufficient to help me understand relevant concepts and do the activity smoothly.	5 (19%)	12 (46%)	6 (23%)	2 (8%)	1 (4%)
SL2: The content and structure of the activity helped me gain confidence in the concepts.	7 (27%)	10 (38%)	9 (35%)	0 (0%)	0 (0%)
SL3: The contents of the activity are relevant to my interests.	5 (19%)	13 (50%)	7 (27%)	1 (4%)	0 (0%)
SL4: It is clear to me how the contents of the activity are related to the targeted concepts.	9 (34%)	16 (61%)	1 (4%)	0 (0%)	0 (0%)
SL5: The activity helped me reinforce relevant concepts.	9 (34%)	16 (61%)	1 (4%)	0 (0%)	0 (0%)
SL6: This activity is an adequate teaching method for the included concepts.	8 (31%)	13 (50%)	3 (11%)	0 (0%)	2 (7%)
Student Engagement					
SE1: I found the activity to be fun/highly engaging (i.e., it does not become monotonous or boring).	7 (27%)	10 (38%)	8 (30%)	1 (4%)	0 (0%)
SE2: Completing the individual tasks/phases of the activity gave me a satisfying feeling of accomplishment.	10 (38%)	7 (27%)	7 (27%)	2 (8%)	0 (0%)
SE3: I was so involved in the activity that I lost track of time.	0 (0%)	6 (23%)	10 (38%)	7 (27%)	3 (12%)
SE4: This activity is appropriately challenging for me.	7 (27%)	12 (46%)	6 (23%)	1 (4%)	0 (0%)
SE5: I would recommend this activity to others.	9 (34%)	11 (42%)	5 (19%)	1 (4%)	0 (0%)
SE6: I prefer learning with this style of activity to other styles that I have experienced.	7 (27%)	13 (50%)	5 (19%)	0 (0%)	1 (4%)



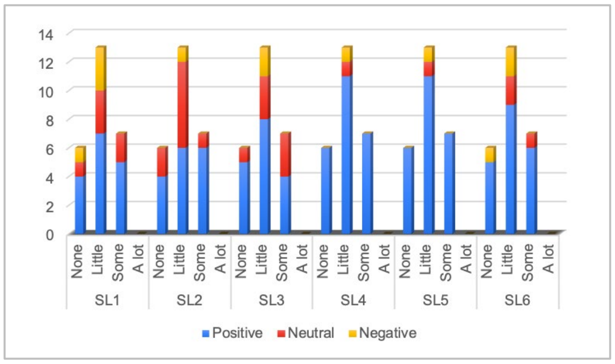
(a) Distribution by Age Group



(b) Distribution by C programming experience

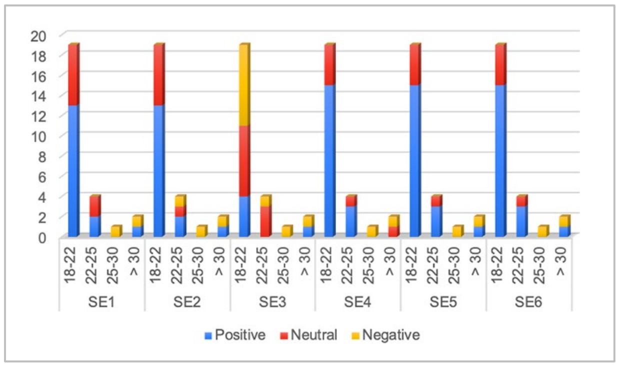


(c) Distribution by Stack Smashing knowledge

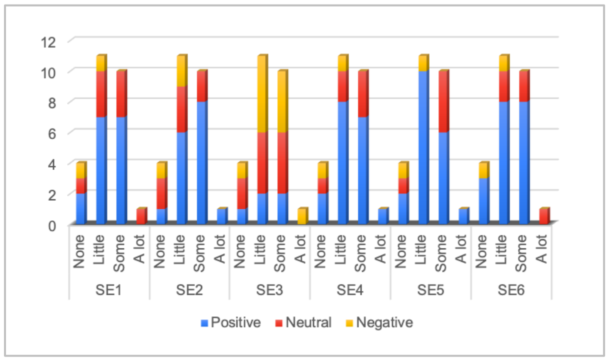


(d) Distribution by Program Visualization tool experience

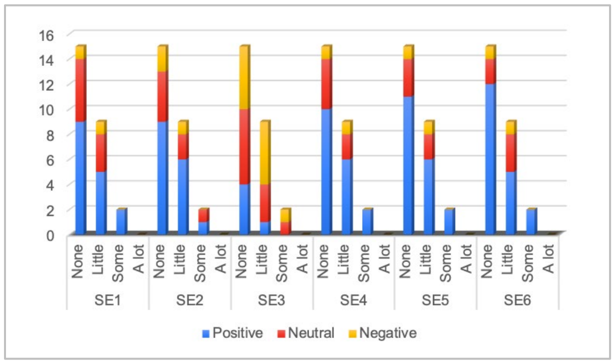
Fig. 1. Responses to Student Learning questions



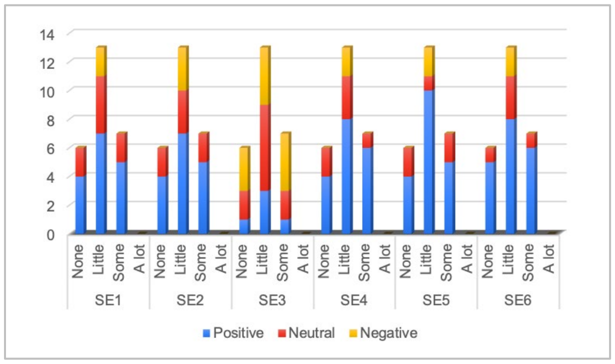
(a) Distribution by Age Group



(b) Distribution by C programming experience



(c) Distribution by Stack Smashing knowledge



(d) Distribution by Program Visualization tool experience

Fig. 2. Responses to Student Engagement questions

3*) SL2 & SL3:* Table I shows that 65 - 69% of students had positive responses to SL2 and SL3, with 27 - 35% neutral responses and 0 - 4% of negative responses. From the SL2 and SL3 bars in Fig. 1a, 1b, 1c and 1d, we see that most of the neutral responses are from students in the 18-22 age group, but with varying levels of C programming experience, stack smashing knowledge and program visualization tool experience. Responses indicate that while the majority of students felt the activity was relevant to them and helped them gain confidence in the concepts, further improvements may be needed to tie the activity better to student interests and needs.

4) SL6: Finally, for SL6, Table I shows that a majority (76%) of the students had positive responses, with 11% of students being neutral and 11% with negative perceptions. The SL6 bars in Fig. 1a, 1b, 1c and 1d show that there are neutral and/or negative responses from students in all age groups and with varying background experience / knowledge. We conclude that perceptions about whether the activity is an adequate teaching method may partly be personal preference, but may be partly related to potential deficiencies that we already noted in our discussion of responses to other Student Learning questions.

C. Student Engagement

To analyze student engagement, we look at data in the Student Engagement section of Table I and Fig. 2a, 2b, 2c and 2d, which show the distribution of responses to the Student Engagement questions for different age groups and varying levels of knowledge / experience with C programming, stack smashing and program visualization tools, respectively.

1) SE1 & SE2: From Table I, we see that over 65% of the students had positive responses to SE1 and SE2, with

26 to 30% of the responses being neutral and only 3 to 7% of them being negative. From the SE1 and SE2 bars in Fig. 2a, 2b, 2c and 2d, we see that positive responses are distributed across most of the groups. Most of the neutral responses are from students in the two younger age groups with varying levels of C programming experience, none or little stack smashing knowledge and varying levels of program visualization tool experience. We also see that the negative responses are from students in older age groups with none to little C programming, stack smashing and program visualization tool knowledge / experience. Overall, this data indicates that DISSAV and the active-learning exercise are engaging in general to students in most groups, but not particularly exciting to any specific groups of students.

2) *SE4 & SE6:* Table I shows that 73 to 76% of the responses to SE4 and SE6 are positive, with 19 to 23% of them being neutral and less than 4% being negative. The bars for SE4 and SE6 in Fig. 2a, 2b, 2c and 2d show that most of the positive responses are from students in younger age groups, but with varying levels of C programming, stack smashing and program visualization tool knowledge / experience. Neutral responses come from students across multiple age groups and varying levels of C programming and program visualization tool experience, but with little to no stack smashing knowledge. Negative responses are mostly from students in older age groups and lower levels of C programming, stack smashing and program visualization tool knowledge / experience. These results suggest that older students with lower levels of prior experience may not have found DISSAV and the active-learning exercise to be appropriately challenging or in a style appealing to them. However, since the numbers of students in most of our groups are small, we do not make any conclusive claim.

3) SE3: From Table I, we see that only 23% of the responses are positive, with 38% of them being neutral and 38% of them being negative. The SE3 bars in Fig. 2a, 2b, 2c and 2d show that positive, neutral and negative responses are distributed across different age groups and backgrounds. This indicates that the DISSAV and the active-learning exercise are generally not engrossing or immersive enough for students to feel that they “lost track of time” (which we note and recognize is a tall order in general for technical learning activities).

4) SE5: Table I shows that almost 77% of students indicated that they would recommend DISSAV and the active-learning exercise, with 19% being neutral and less than 4% being negative. Positive, neutral and negative responses are spread out across students in all age groups except 25-30, with varying levels of C programming, stack smashing and program visualization tool knowledge / experience. Overall, this is a very encouraging result. It suggests that, while there may be some improvements needed to DISSAV and the active-learning exercise, they are solid resources that a majority of students would recommend them to others.

D. Open-ended responses

In addition to the Likert-scale questions, we asked two free-response questions about DISSAV and the active-learning exercise. First, we asked students to “list two strong aspects of the activity”. We find three aspects of the tool that are common among several student answers. The most common aspect that students like is the visual representation of different components. Another common, strong aspect is the ease of navigation throughout the tool. Finally, students also state that they find the tool engaging. Then, we asked students to “give two suggestions to improve the activity” and find two suggestions that are common among several student answers. The most common suggestion is to provide more explanation or hints for the process of simulating a stack smashing attack using DISSAV, especially constructing an attack payload. While explanations / hints are provided in the activity, some students feel that more detail would make the activity smoother. The other common suggestion is that the tool needs User Interface improvements, specifically better window scaling for different laptop and screen sizes.

V. DISCUSSION

A. Results

Overall, the results of our data analysis are encouraging. On an average, we find that over 80% of student responses to User Interface questions are positive, indicating that DISSAV has a well-designed interface. An average of over 66% of the responses to Student Learning questions are positive, with over 96% of positive responses to questions related to the relevance to and reinforcement of targeted concepts. This demonstrates that DISSAV and the active-learning exercise have a high potential of improving student learning in the complex and important topic of stack smashing, thus answering research question R1. We find that an average of over 63% of the responses to Student Engagement questions are positive, with that number going up to over 71% if we remove the question that asks students if they were so involved in the activity that they lost track of time (SE3). This indicates that, with some improvements, DISSAV and the active-learning exercise can effectively engage students in the learning experience, thus answering research question R2. There are differences in the distribution of responses from students in different age groups and students with different levels of knowledge / experience in C programming, stack smashing and program visualization tools. However, from our current data, we do not observe specific benefits or shortcomings that our resources cause to specific groups. The sizes of some of the groups are too small for us to determine whether the observed differences in responses are significant and related to the demographics or whether they are simply differences between the perceptions of individual students, unrelated to the groups to which they belong. Thus, we are unable to conclusively answer research question R3.

B. Challenges and Limitations

We faced multiple challenges related to this study. First, due to the fact that Fall 2021 was the first semester back on campus after the COVID-19 pandemic, some students were apprehensive about being back. This made it more difficult to engage them in classroom activities. Second, stack smashing requires a vast amount of background information, making it difficult to create activities that are in-depth, yet short enough to be completed in a class period while also being engaging. Lastly, we faced a significant challenge in encouraging students to complete and submit the voluntary student survey because it had no associated grade component. Specifically, only 26 out of 104 students completed the voluntary survey and consented to have their responses collected and analyzed. This prevented a more substantial statistical analysis of responses to the student survey and limited us to general observations.

VI. RELATED WORKS

Sasano [18] presents a tool for visualizing buffer overflows in C programs and detecting return address overwriting. While the tool targets novice C programmers, the authors suggest that it may be useful for experienced programmers as well. The paper focuses on the presentation of the tool and does not provide an evaluation of the tool.

Zhang et al. [19] develop and evaluate a web-based interactive visualization tool to teach buffer overflow concepts. The authors evaluate the effectiveness of this tool by conducting a study that uses a pre-test / post-test and focus group discussions in two small classes (one undergraduate and one graduate). The study finds that using such a tool to learn about buffer overflow concepts improved student motivation / engagement and made it easier for them to understand the topic.

Walker et al. [20] present and report their experiences with a program analysis and visualization tool designed to help students to visualize a program's address space in order to eventually understand security issues within C programs. The authors evaluate the tool in a junior level systems programming courses, via a pre-test / post-test and an evaluation form. The study finds a significant improvement in post-test scores when compared to pre-test scores and reports generally positive student feedback.

Simple Machine Simulator (SMS) [13] is an interactive visualization tool that demonstrates stack frame and buffer overflow concepts and their effects on process memory in a hands-on lab environment. While the paper does not present a detailed evaluation of the tool, the authors indicate that the SMS tool was used successfully in their computer security course and specifically note that non-Computer Science majors were able to complete and understand the concepts through the use of SMS.

Unlike DISSAV, none of the above security related visualization tools present the ability to simulate a stack smashing attack through a customizable C program, dynamically created payload and an interactive call stack visualization.

VII. CONCLUSIONS

In this paper, we present the deployment and analysis of a program visualization tool, DISSAV, and an accompanying active-learning exercise to teach stack smashing, a key software security attack. The visualization tool and active-learning exercise work together to guide students through various challenges presented by stack smashing attacks.

We present the results of deploying DISSAV and the accompanying exercise in an undergraduate cybersecurity course that introduces a broad range of security topics. We report on the results of simple study that collects and evaluates student perceptions on the user interface of DISSAV and the effectiveness of the two resources in improving student learning and engagement. The study finds that the majority of students find DISSAV and the active-learning exercise engaging and that the resources improve their perceived learning.

VIII. ONGOING AND FUTURE WORKS

In recent work, we transformed a sequence of activities that aim to teach students about strings and buffer overflows in C, process memory layout, and stack smashing attacks into shorter, more engaging, guided learning style activities. We expect that these activities will help students develop a more solid understanding of foundational knowledge before being exposed to DISSAV. We have also enhanced the active-learning exercise accompanying DISSAV to include more explanations and hints to help students work through the tool, construct an attack payload and simulate a stack smashing attack. We deployed the guided learning activities and the enhanced active-learning exercise in Fall 2022 and collected data. We plan to analyze this data in future work. We also plan to make DISSAV's user interface more responsive so that it adapts to different screen sizes and devices. Overall, we expect that these changes will improve student learning and engagement further. To gain insights into actual improvements in student learning, we plan to conduct a comparative analysis of student performance on assessments related to stack smashing from semesters prior to the inclusion of the guided learning style activities and DISSAV and semesters during which they were included.

REFERENCES

[1] A. One. Smashing the stack for fun and profit. Accessed on 12.12.2022. [Online]. Available: https://inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf

[2] C. Hritcu. Control hijacking attacks. Accessed on 12.12.2022. [Online]. Available: https://catalin-hritcu.github.io/talks/04-control-hijacking-attacks.pdf

[3] L.-H. Chen, F.-H. Hsu, C.-H. Huang, C.-W. Ou, C.-J. Lin, and S.-C. Liu, "A robust kernel-based solution to control-hijacking buffer overflow attacks," *J. Inf. Sci. Eng.*, vol. 27, pp. 869–890, 05 2011.

[4] M. Khurana, R. Yadav, and M. Kumari, "Buffer overflow and SQL injection: To remotely attack and access information," *Cyber Security*, pp. 301–313, 2018.

[5] J. Xu, Z. Kalbarczyk, S. Patel, and R. Iyer, "Architecture support for defending against buffer overflow attacks," *Coordinated Science Laboratory Report no. UILU-ENG-02-2205, CRHC-02-05*, 2002.

[6] V. English, I. Obaidat, and M. Sridhar, "Exploiting memory corruption vulnerabilities in connman for IoT devices," *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 247–255, 2019.

[7] A. Mohanty, I. Obaidat, F. Yilmaz, and M. Sridhar, "Control-hijacking vulnerabilities in IoT firmware: A brief survey," *Proceedings of the 1st International Workshop on Security and Privacy for the Internet-of-Things*, 04 2018.

[8] L. Vaas. Pulse secure VPNs get quick fix for critical RCE. Accessed on 12.12.2022. [Online]. Available: https://threatpost.com/pulse-secure-vpns-critical-rce/166437/

[9] S. Gatlan. Foxit reader bug lets attackers run malicious code via PDFs. Accessed on 12.12.2022. [On-line]. Available: https://www.bleepingcomputer.com/news/security/foxit-reader-bug-lets-attackers-run-malicious-code-via-pdfs/

[10] I. Murphy. Do you suffer from patch lag? Accessed on 12.12.2022. [Online]. Available: https://www.enterprisetimes.co.uk/2021/03/10/do-you-suffer-from-patch-lag/

[11] D. Wang, M. Jiang, R. Chang, Y. Zhou, B. Hou, X. Luo, L. Wu, and K. Ren, "A measurement study on the (in)security of End-of-Life (EoL) embedded devices," *arXiv preprint arXiv:2105.14298*, 2021.

[12] L. Caviglione, S. Wendzel, A. Mileva, and S. Vrhovec, "Guest editorial: Multidisciplinary solutions to modern cybersecurity challenges," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 12, no. 4, pp. 1–3, December 2021.

[13] D. Schweitzer and J. Boleng, "A simple machine simulator for teaching stack frames," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 2010, pp. 361–365.

[14] E. Heinsen and C. McDonald, "Program visualization and explanation for novice C programmers," *Proceedings of the Sixteenth Australasian Computing Education Conference*, vol. 148, pp. 51–57, 2014.

[15] D. Budny, L. Lund, J. Vipperman, and J. Patzer, "Four steps to teaching c programming," in *32nd Annual Frontiers in Education*, vol. 2. IEEE, 2002.

[16] D. Radošević, T. Orehovački, and A. Lovrenčić, "New approaches and tools in teaching programming," in *Radošević, D., Orehovački, T., Lovrenčić, A: "New Approaches and Tools in Teaching Programming", Central European Conference on Information and Intelligent Systems, CECIIS*, 2009.

[17] E. Akeyson, H. Ramaprasad, and M. Sridhar, "DISSAV: A Dynamic, Interactive Stack-Smashing Attack Visualization tool," *2022 Journal of The Colloquium for Information Systems Security Education*, vol. 9, no. 1, 2022.

[18] I. Sasano, "A tool for visualizing buffer overflow with detecting return address overwriting," *EAI Endorsed Transactions on Self-Adaptive Systems*, vol. 2, no. 5, 2016.

[19] J. Zhang, X. Yuan, J. Johnson, J. Xu, and M. Vanamala, "Developing and assessing a web-based interactive visualization tool to teach buffer overflow concepts," *IEEE Frontiers in Education Conference*, pp. 1–7, 2020.

[20] J. Walker, M. Wang, S. Carr, J. Mayo, and C.-K. Shene, "A system for visualizing the process address space in the context of teaching secure coding in C," *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pp. 1033–1039, 2020.