Security Injections 2.0: Increasing Ability to Apply Secure Coding Knowledge using Segmented and Interactive Modules in CS0

Sagar Raina
Towson University
7800 York Rd.
Towson, MD, USA - 21252
443-824-7220
sraina1@students.towson.edu

Siddharth Kaza
Towson University
7800 York Rd.
Towson, MD, USA - 21252
410-704-6310
skaza@towson.edu

Blair Taylor
Towson University
7800 York Rd.
Towson, MD, USA - 21252
410-704-4560
btaylor@towson.edu

ABSTRACT

Student skipping content is common in traditional learning modules that present a large amount of content in a linear format. This can lead to lower student engagement, and may yield poor learning. In this paper, we compare student learning between enhanced learning modules (2.0) and traditional modules (1.0) aimed at providing knowledge on secure coding to students in lower-level programming courses. We discuss the results of a quasi-experiment across two sections of CS0. The study compares students' secure coding awareness, general software security awareness and students' ability to identify security vulnerabilities (integer overflow, input validation and buffer overflow) in three separate code segments. A total of 53 students participated in the study. While results indicate significant improvement in secure coding and general software security awareness in both 1.0 and 2.0 modules, students using 2.0 modules performed significantly better than students using 1.0 modules in applying secure coding knowledge by identifying security vulnerabilities in code segments.

Keywords

Security Injections; integer overflow; buffer overflow; input validation; cs0; cs1; cs2; learning sciences; interactive learning modules; instant-feedback; auto-grading.

1. INTRODUCTION

Cybersecurity education is a crucial component in addressing the cybersecurity crisis [15]. Several cybersecurity learning materials, including module-based, have been developed across United States [2]. Traditional learning modules that present a large amount of content in a linear format may lead to content skipping, lower engagement and poor learning [11, 14]. The original Security Injections @Towson cybersecurity modules, though effective, followed a traditional linear format [15]. One of the issues that was observed by instructors was that students tended to skip content and proceed directly to lab exercises. To address this issue, in our previous study [13], we proposed to incorporate e-learning design principles of segmentation and interactivity and hypothesized – 1) segmentation of content will reduce content skipping, 2) reduced content skipping will increase learning, 3) interactivity will increase engagement and 4) an increase in engagement will increase learning [11, 12, 14]. The results indicated that the enhanced modules significantly increased the student engagement than the traditional modules [13].

SIGCSE '16, March 02–05, 2016, Memphis, TN, USA

DOI: http://dx.doi.org/10.1145/2839509.2844609

In this paper, we extend our study to examine differences in student learning on concept retention and ability to apply secure coding knowledge between enhanced - (2.0) and traditional (1.0) modules using Security Injections @Towson cybersecurity modules. First, we describe the theoretical framework for segmentation and interactivity; second, we discuss learning differences between interactive and non-interactive systems; third, we describe the enhanced (2.0) modules; and fourth, we discuss the methodology and results of a quasi-experiment across two sections of CS0 that compares students' secure coding awareness, general software security awareness, and students' ability to apply the knowledge they gained by identifying security vulnerabilities (integer overflow, input validation and buffer overflow) in three separate code segments. Previous research has shown that while retention of knowledge is not affected by the interactive nature of the system [5, 6, 9, 17], students perform significantly better on application of knowledge using interactive systems [5, 17]. Based on this we examine the following research questions:

RQ1: Will the enhanced (2.0) and traditional (1.0) modules show the same level of learning for questions related to retention of the material?

RQ2: Will enhanced (2.0) modules show significantly higher learning for identifying security vulnerabilities in code segments (application of knowledge) than traditional (1.0) modules?

2. LITERATURE REVIEW

In this section, we briefly discuss – 1) the theoretical framework for incorporating segmentation and interactivity in traditional linear modules, 2) review previous literature that compares learning in interactive and non-interactive systems.

2.1 Segmentation and Interactivity

Traditional linear web-based modules that present large amount of text on a single page may lead to content skipping and skimming, and lower engagement [11, 14]. Research suggests that skipping in a large web-based hypertext pages could be due to – 1) flexibility to click any of the hyperlinks that links within or

outside the page to gain knowledge and thus loosing context of the original text [3, 11], 2) reading strategy adopted by readers which determines what to read and what to skim [7, 8]. This may lead readers to lose important information, and result in shallow reading, less concentration and attention towards content [4, 8]. Overall, skipping of the content may lead to poor learning [14]. In addition, lower engagement in the traditional linear modules could be due to the less interactive activities, which may also result in less learning [12].

To overcome the issues in traditional linear modules, we proposed to incorporate e-learning design principles of segmentation and interactivity [13]. Segmentation implies breaking large content into smaller chunks and present one chunk at a time on a single screen. Segmentation makes processing, retention and recalling of information easier [1, 10]. In addition, to increase engagement with the module interface, research suggests increasing user-system interactivity [12]. Interactivity in e-learning is the "responsiveness to the learner's actions during learning." We proposed to increase interactivity with dialoguing and controlling. The process of a learner answering a question and receiving feedback on his/her input is referred to as dialoguing. Dialoguing improves learning [16], as learners can relate feedback to the current content. Controlling implies that the learner can determine the pace of the presentation. Controlling helps students learn better by allowing them to process information at their own pace.

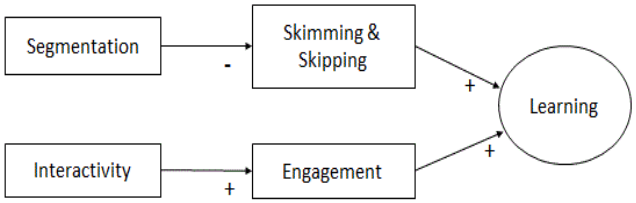


Figure 1. Literature suggests that segmentation and interactivity in modules may increase learning

Overall, segmentation breaks large content into smaller chunks and presents them one at a time, which may result in less reading and less skipping of content. Less skipping of content may lead to increased learning. Interactivity (dialoging and controlling) on segmented chunks leads to engagement and enforces learning (see Figure.1).

2.2 Interactive versus Non-interactive Learning Systems

Hattie [6] state that any kind of educational intervention has a positive effect on student achievement. Then, do interactive and non-interactive systems with similar content will have the same learning effect? A similar study conducted by Evans and Gibbons [5] showed that both interactive and non-interactive systems with same content have same learning when assessment questions examine retention or recall. But, interactive systems performed significantly better than non-interactive system when assessment involved problem-solving.

Another study by Wang et. al. [17] examined impact of animation interactivity on novices' learning of introductory statistics. The study comprised of three groups – 1) static group – provided with static material, 2) simple animation group – animation with input manipulation, and 3) practice group – animation with practice and feedback. The results showed animation interactivity significantly improved students' understanding and lower level applying.

A study by Mayer et. al. [9] found that students performed significantly better on problem solving transfer test due to the interactive feedback provided by the system compared to non-interactive version.

Therefore, based on the literature on interactive and non-interactive systems with same content, we can conclude that interactive systems may not show significant improvement on recall or retention assessments but could show significant improvement on problem-solving assessments where students have to apply their understanding of the knowledge.

3. INCORPORATING SEGMENTATION AND INTERACTIVITY

The original (1.0) modules were developed on the cognitive learning principles of Bloom's taxonomy and adopt a uniform structure. Each module begins with a background section to describe the problem with examples, followed by a "Code Responsibly" section (includes methods to avoid security issues), a laboratory assignment with a security checklist, and discussion questions. The module structure is designed to help students to first *understand* the problem through the background and code responsibly sections, *remember* it through the laboratory assignments, *evaluate* it through checklists, and *apply* the concepts learned through discussion questions. This ensures the implementation of active learning in Security Injection modules.

Using the theoretical framework discussed in section 2.1, we enhanced the 1.0 modules. We implemented segmentation by breaking up the module content per section (background, code responsibly, laboratory assignment, discussion questions) and present each section, one at a time, on the screen. In this fashion, the reader views a small amount of content at a time. We implemented dialoguing using formative assessment with a set of checkpoint questions, both multiple choice and constructed response that include feedback. We selected multiple choice questions, as they can be used to infuse both surface and deep learning, whereas constructed response are known for infusing deep learning among learners. We implement immediate elaborate feedback with the correct answers after students attempt the questions.

Each section in a module is auto-graded using built-in functionality for text and multiple-choice questions. In the background and code responsibly sections, students are required to go through the content and answer a set of checkpoint questions. Each question provides immediate feedback on submit (see Figure 2). The student cannot advance to the next section until all questions are answered correctly. In the Laboratory Assignment and Discussion Question, students answer text-based, multiple-choice questions, and identify vulnerabilities based on a security checklist (see Figure 3). These are also auto-graded.

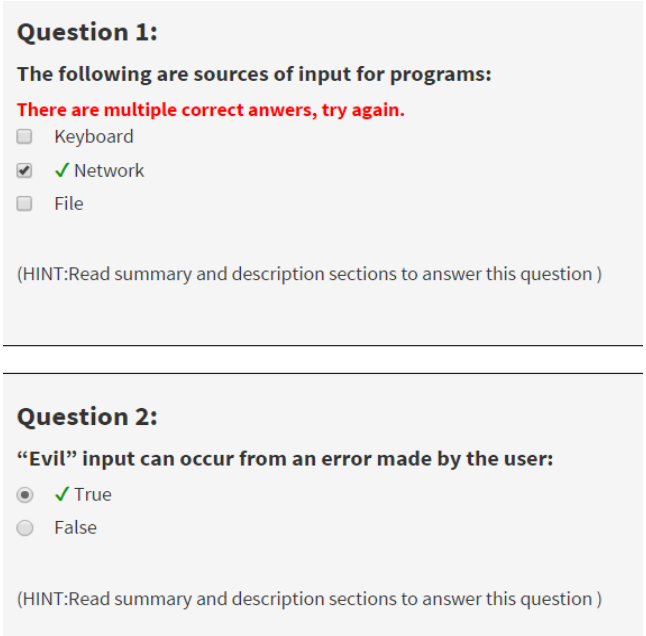
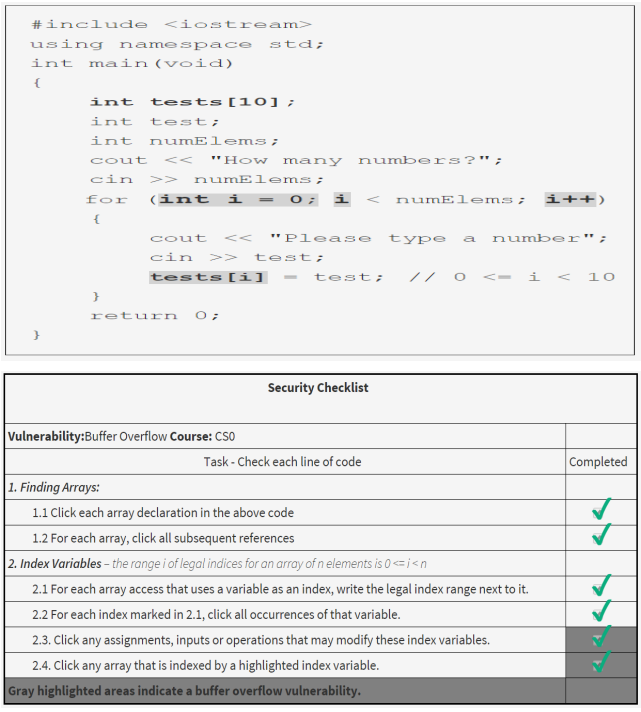


Figure 2. Instant-feedback



```
#include <iostream>
using namespace std;
int main(void)
{
     int tests[10];
     int test;
     int numElems;
     cout << "How many numbers?";
     cin >> numElems;
     for (int i = 0; i < numElems; i++)
     {
          cout << "Please type a number";
          cin >> test;
          tests[i] = test;  // 0 <= i < 10
     }
     return 0;
}
```

Security Checklist	
Vulnerability: Buffer Overflow **Course:** CS0	
Task - Check each line of code	Completed
1. Finding Arrays:	
1.1 Click each array declaration in the above code	✓
1.2 For each array, click all subsequent references	✓
2. Index Variables – the range i of legal indices for an array of n elements is 0 <= i < n	
2.1 For each array access that uses a variable as an index, write the legal index range next to it.	✓
2.2 For each index marked in 2.1, click all occurrences of that variable.	✓
2.3. Click any assignments, inputs or operations that may modify these index variables.	✓
2.4. Click any array that is indexed by a highlighted index variable.	✓
Gray highlighted areas indicate a buffer overflow vulnerability.	

Figure 3. Auto-graded security checklist

4. PILOT STUDY

4.1 Methodology

A quasi-experiment was conducted in two sections of a CS0 course (programming logic using C++) in spring 2015 at a large public university, using a pretest – posttest control group design. Both the sections were taught by the same instructor. One of the sections used the 1.0 version (control group) and the other used 2.0 (treatment group). The study was conducted during the laboratory sessions, which were at different times for each section (11-11:50 AM (control), 12-12:50 PM (treatment)). Three modules - integer error, input validation and buffer overflow - were introduced, in that order, with approximately four weeks between the interventions. Both groups were administered a pre-survey at the beginning and a post-survey at the end of the semester. A total of 53 (26 in the treatment group and 27 in the control group) students participated in the study. See Table 1. for complete student demographics. As can be seen, both groups were comparable.

Table 1. Student Demographics

	Control		Treatment	
	n	**%**	**n**	**%**
Gender				
Male	16	61.5	17	63.0
Female	10	38.5	10	37.0
Age				
20 years or younger	18	69.2	17	63.0
21-25 years	7	26.9	9	33.0
26-30 years	1	3.8	1	3.7
Ethnicity				
White	15	57.7	17	63.0
Black	4	15.4	6	22.2
Hispanic	1	3.8	2	7.4
Asian	5	19.2	2	7.4
Other	1	3.8	0	0.0
Student Standing				
Freshman	7	26.9	10	37.0
Sophomore	10	38.5	7	25.7
Junior	8	30.8	8	29.6
Senior	1	3.8	2	7.4
Major				
Computer Science	14	57.7	11	40.7
Non-Computer Science	12	42.3	16	59.3

4.1.1 Instruments

The survey instruments (both pre-survey and post-survey) used in this study were derived from previous security injections studies [15]. Both the pre-survey and post-survey include multiple choice questions related to student demographics, secure coding awareness and general software security awareness. Secure coding awareness include 5 questions - 2 integer overflow, 2 input validation and 1 buffer overflow; and, 4 questions for general software security awareness. In addition, 3 code segments were added to the post-survey to assess students' ability to apply secure coding knowledge. The code segments were developed by the senior instructors teaching CS0. The students were to identify the potential security vulnerability in the code segments. See Table 3 for security awareness questions and Table 2 for code segments (ability to apply).

Table 2. Code Segments (Ability to apply)

Identify the potential security issues in the following code segment: (Check all that apply)

Code Segment 1
`float price;`
`float totalPrice;`
`cout << "Enter Price" << endl;`
`cin >> price;`
`totalPrice = price + price*.06;`
 	
Code Segment 2
`//assume i < INT_MAX and j < INT_MAX`
`int calc (int i, int j) )`
`{`
`int result = i * j;`
`return result;`
`}`
 	
Code Segment 3
`//assume n < INT_MAX`
`void input(float temperatures[], int n)`
`{`
`for (int i = 0; i < n; i = i + 1)`
`{`
`cout << temperatures[i] << endl;`
`}`
`}`

Table 3. Survey questions (Awareness)

Secure-coding Awareness
Integer Overflow occurs ..
Integer Overflow is caused by ..
Invalid input can come from ..
Which of the following should your well designed program do before processing user input?
Which programming mistake is one of the major vulnerabilities in today applications?
Software-security Awareness
What are the possible consequences of insufficient computer security?
Security Software and Software Security are the same:
When developing secure systems, where does security fit in?
Software security vulnerabilities are the result of software bugs and flaws:
Your code is completely secure if:

Based on the pre-survey and the post-survey scores, we proposed the following set of hypotheses to compare Security Injections 1.0 (control group) and Security Injections 2.0 (treatment group) on the following dependent variables: secure coding awareness, general software security awareness, and ability to apply secure coding knowledge.

H1: The post-survey scores for secure coding awareness (measuring retention) will be significantly higher than the pre-survey scores for secure coding awareness in both control and treatment groups.
Rationale – As discussed in section 2.2, any kind of educational intervention has a positive effect on student achievement. In addition, interactive and non- interactive systems with same content will have the same learning on recall or retention assessments.

H2: The post-survey scores for secure coding awareness (measuring retention) for the treatment and the control group will not be significantly different.
Rationale – As discussed in section 2.2, both interactive and non-interactive systems with same content will have the same learning on recall or retention assessments. Secure coding awareness assesses retention of integer overflow, input validation and buffer overflow knowledge.

H3: The post-survey scores for general software security awareness (measuring retention) will be significantly higher than the pre-survey scores for general software security awareness in both control and treatment group.
Rationale – Same as H1

H4: The post-survey scores for general software security awareness (measuring retention) for the treatment and the control group will not be significantly different.
Rationale – As discussed in section 2.2, both interactive and non-interactive systems with the same content will have same learning for recall or retention assessments. General software security awareness assesses retention of general software security knowledge.

H5: The scores for ability to apply secure coding knowledge in the treatment group will be significantly higher than the control group.

Rationale- As discussed in section 2.2, interactive systems will show significantly higher learning on problem-solving (ability to apply) assessments than the non-interactive systems with same content. The students apply their secure coding knowledge to identify security vulnerability in the code segments.

H1 - H4 addresses RQ1 and, **H5** addresses RQ2

4.2 Results

A total of 53 (26 in the control and 27 in the treatment) students, including 33 males and 20 females, participated in the study and completed the post-survey. 21 (11 males and 10 females) students in the control group and 24 (14 males and 10 females) in the treatment completed both pre and post survey.

The scores for each category in the survey were calculated based on the correct answers out of - 5 for secure coding awareness (integer overflow (2), input validation (2), buffer overflow (1)), 5 for general software security awareness and 9 for ability to apply secure coding knowledge on code segments (integer overflow (3), input validation (3) and, buffer overflow (3)).

H1 and H3 were tested using Wilcoxson-Signed-Ranks non-parametric test to compare the mean rank of scores between pre-survey and post-survey scores. We picked the non-parametric test because – 1) the Kolmogorov-Smirnov and Shapiro-Wilk test showed that the scores are not normally distributed ($p < 0.05$), and 2) the groups (pre and post) were related samples. H2, H4 and H5 were tested using Mann-Whitney non-parametric test to compare the mean rank of the scores in two groups (control and treatment). We picked non-parametric test because – 1) n for the group were not equal, 2) Kolmogorov-Smirnov and Shapiro-Wilk test showed that the scores are not normally distributed ($p < 0.05$), and 3) the two groups were independent samples.

Comparison for pre-survey and post-survey scores for secure coding awareness in treatment and control group.
In the treatment group, the average score for pre-survey (1.5) and the average score for post-survey (4.0) were found to be statistically significant at the 95% level ($p < 0.05$, $z = -4.02$). In the control group, the average score for pre-survey (1.76) and the average score for post-survey (3.81) were found to be statistically significant at the 95% level ($p < 0.05$, $z = -3.78$). This implies that the use of both Security Injections 1.0 and 2.0 significantly increased the secure coding awareness (measuring retention) among the students in the post-survey compared to the pre-survey. In addition, this verifies that any kind of educational intervention has a positive effect on student achievement, and, interactive and non- interactive systems with same content will have the same learning on recall or retention assessments. This leads us to accept H1 (see Figure 4).

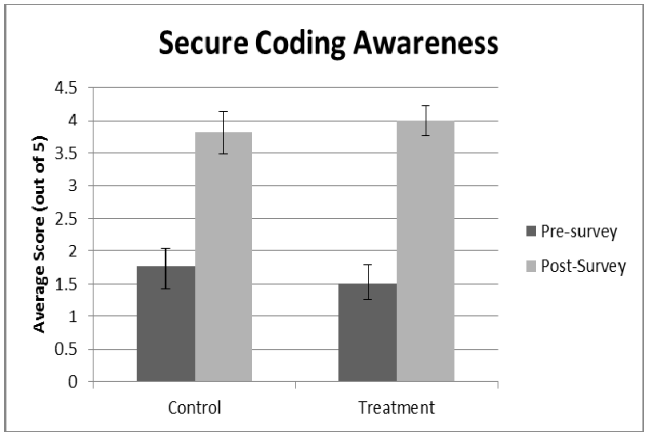


Figure 4. Pre-survey and post-survey scores in the control and treatment groups for security-coding awareness

Comparison of post-survey scores for secure coding awareness in treatment and control groups
In the post-survey, no significant differences were found between the average scores for the treatment group (4.0) and the control group (3.81). This verifies that interactive and non-interactive systems with same content have same learning on recall or retention assessments. This leads us to accept H2 (see Figure 4).

Comparison of pre-survey and post-survey scores for general software security awareness in treatment and control group.
In the treatment group, the average score for the post-survey (4.21) was significantly higher at the 95% level ($p < 0.05$, $z = -3.056$) than the average score for the pre-survey (3.25). In the control group, the average score for the post-survey (4.14) was also significantly higher at the 95% level ($p < 0.05$, $z = -2.20$) than the average score for the pre-survey (3.29). This implies that the use of both Security Injections 1.0 and 2.0 significantly increased the general software security awareness among the students in the post-survey compared to the pre-survey. In addition, this verifies that any kind of educational intervention has a positive effect on student achievement, and, interactive and non-interactive systems with same content will have the same learning on recall or retention assessments. This leads us to accept H3 (see Figure 5).

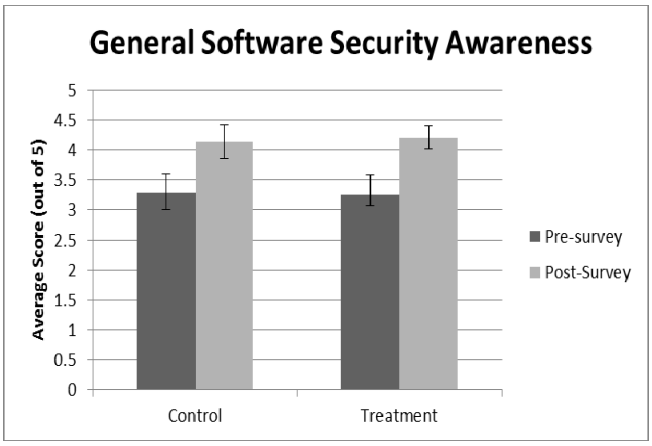


Figure 5. Pre-survey and post-survey scores in the control and treatment groups for general software security awareness

Comparison of post-survey scores for general software security awareness in treatment and control groups
In the post-survey, no statistically significant differences were found between the average scores for the treatment group (4.21) and the control group (4.14). This verifies that interactive and non-interactive systems with same content have same learning on recall or retention assessments. This leads us to accept H4. (see Figure 5)

Comparison of post-survey scores for ability to apply secure coding knowledge in treatment and control groups
In the post-survey, the average score for the treatment group (5.59) was found significantly higher at 90% level ($p = 0.07 < 0.10$, $z = -1.80$) than the average score for the control group (4.27). The students who use enhanced (2.0) modules performed significantly better in identifying security vulnerabilities in three separate code segments than the students who use 1.0. This verifies interactive systems show significantly higher learning on problem-solving (ability to apply) assessments than the non-interactive systems with same content. This leads us to accept H5 (see Figure 6).

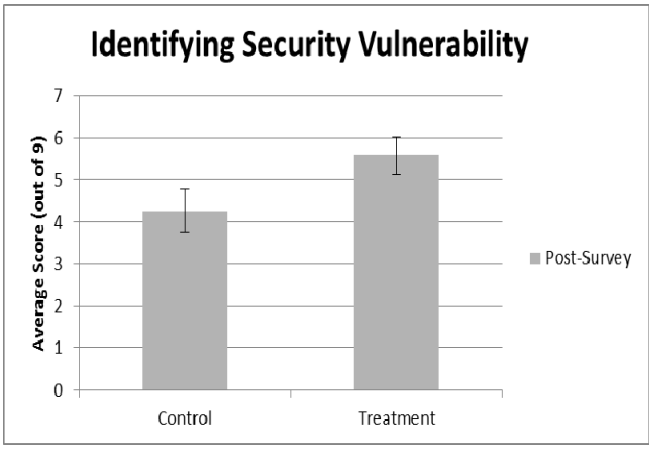


Figure 6. Post-survey scores in the control and treatment groups for ability to identify security vulnerability in three code segments

5. CONCLUSION

In this paper, we extended our previous study to compare student learning between traditional (1.0) and enhanced (2.0) secure coding modules. Traditional modules follow linear format with non-interactive content whereas enhanced module content were segmented and interactive using checkpoint questions, and auto-graded security checklist with instant-feedback. We conducted a quasi-experiment to compare students' secure coding awareness, general software security awareness and students' ability to identify security vulnerabilities (integer overflow, input validation and buffer overflow) in three separate code segments.

Results showed significant improvement in secure coding and general software security awareness (measuring retention) in both 1.0 and 2.0. Additionally, students using segmented and interactive modules performed significantly better in identifying security vulnerabilities in code segments than traditional modules.

In future, we plan to explore the results based on demographics. In addition, we plan to examine another component of our research framework - whether segmentation leads to less skipping and skimming. We plan a usability study in fall 2015 and spring 2016 to record students reading pattern in both 1.0 and 2.0 versions using eye-tracking software.

6. ACKNOWLEDGMENTS

This project is partially supported by NSF DUE-1241738 and 0817267.

7. REFERENCES

[1] Clark, R.C. and Mayer, R.E. 2011. *e-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning (Google eBook)*. John Wiley & Sons.

[2] Curriculum Resources - Teaching Tools for Educators: *http://niccs.us-cert.gov/education/curriculum-resources*.

[3] DeStefano, D. and LeFevre, J.-A. 2007. Cognitive load in hypertext reading: A review. *Computers in Human Behavior*. 23, 3 (May 2007), 1616–1641.

[4] Duggan, G.B. and Payne, S.J. 2011. Skim Reading by Satisficing: Evidence from Eye Tracking. *CHI 2011* (Vancouver, BC, Cananda, 2011).

[5] Evans, C. and Gibbons, N.J. 2007. The interactivity effect in multimedia learning. *Computers & Education*. 49, 4 (Dec. 2007), 1147–1160.

[6] Hattie, J. 2013. *Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement*. Routledge.

[7] Lawless, K.A. et al. 2003. Knowledge, Interest, Recall and Navigation: A Look at Hypertext Processing. *Journal of Literacy Research*. 35, (Jan. 2003), 911–934.

[8] Liu, Z. 2005. Reading behavior in the digital environment: Changes in reading behavior over the past ten years. *Journal of Documentation*. 61, 6 (2005), 700–712.

[9] Mayer, R.E. et al. Multimedia Learning in an Interactive Self-Explaining Environment: What Works in the Design of Agent-Based Microworlds?

[10] Moreno, R. and Mayer, R. 2007. Interactive Multimodal Learning Environments. *Educational Psychology Review*. 19, 3 (Jun. 2007), 309–326.

[11] Protopsaltis, A. and Bouk, V. 2005. Towards a Hypertext Reading/Comprehension Model. *SIGDOC'05* (2005).

[12] Quinn, C.N. 2005. *Engaging Learning: Designing e-Learning Simulation Games*. John Wiley & Sons.

[13] Raina, S. et al. 2015. Security Injections 2.0: Increasing Engagement and Faculty Adoption using Enhanced Secure Coding Modules for Lower-level Programming Courses. *9th World Conference on Information Security Education* (Hamburg, Germany, May 2015).

[14] Rudestam, K.E. and Schoenholtz-Read, J. 2010. *Handbook of Online Learning*. SAGE Publications.

[15] Taylor, B. and Kaza, S. 2011. Security injections: modules to help students remember, understand, and apply secure coding techniques. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11* (New York, New York, USA, 2011), 3.

[16] Thalheimer, W. 2008. *Providing Learners with Feedback—Part 1: Research-based recommendations for training, education, and e-learning*.

[17] Wang, P.-Y. et al. 2011. The impact of animation interactivity on novices' learning of introductory statistics. *Computers & Education*. 56, 1 (Jan. 2011), 300–311.