

Explaining the Buffer Overflow Problem: Instructional Design and Evaluation in Information Security Education

Susan L. Gerhart
Embry-Riddle Aeronautical University
gerharts@erau.edu

Jan G. Hogle
Big Cat Communications
jan@twinpinefarm.com

Jedidiah R. Crandall
U. C. Davis
crandall@cs.ucdavis.edu

Abstract

Today's headlines read: "Buffer overflow in vendor's product allows intruders to take over computer!" What can software engineering education do about this situation? This paper describes Java applet demonstrations appropriate for traditional undergraduate courses designed to drive home the problem: why buffer overflows occur, how overflows open the door to attackers, and why certain defense mechanisms should be used. Supporting contextual and instructional materials were developed in conjunction with the Java applets. Continuing evaluation strategies and observations indicate the demonstrations provide immediate and powerful impact that supports long-term learning, continuing interest in security, and awareness of professional issues. Applet demos and supporting materials are available at the project Web site [<http://nsfsecurity.pr.erau.edu>].

1. Introduction

We briefly describe the motivation for this work, the [buffer overflow problem](#) and its roots in software engineering, a [demonstration-centered portfolio of training materials](#), and our experience and [evaluation](#) to date. A grant (<http://nsfsecurity.pr.erau.edu>) under the Capacity Building Track of the NSF Federal Cyber Service program links software engineering, global intelligence, and educational technology expertise to develop curriculum modules appropriate for undergraduates. Initial modules are targeted to: (1) maximally increase the current security content of the Embry-Riddle curriculum, without perturbing course sequencing; (2) assess interest from students and (3) increase faculty competence and involvement. Additional module criteria include: (a) interactive using current computing technology; (b) applying standard methodology for designing and evaluating instructional modules; and (c) dissemination to other training sites.

The Buffer Overflow problem was an obvious starting point, given the notoriety and persistence of the problem. "Buffer overflow" is vying with Y2K for top billing

among programming bugs notable to the public [1]. The prominence is dramatic, e.g. measured by query responses at security portals: 1035 in the MITRE CVE <http://cve.mitre.org>, 721 at CERT <http://www.cert.org>, and 928 at InfoSysSec <http://www.infosyssec.com>.

2. The Buffer Overflow Problem

A "buffer overflow" is said to occur when a pointer (as in C) goes out of range to access memory beyond the buffer. Frequently, the buffer is part of a stack frame, or activation record, associated with other defined variables (such as PasswdOK) and locations for change of control. Determining the compiler strategy for memory layout allows an attacker to define a string that, when placed in the buffer, overflows to achieve the attacker's goal, e.g. gaining root privileges or changing a PasswdOK variable.

While excellent web explanations are readily accessible [2,3], few traditional textbooks directly address the problem. Inappropriate use of pointers and library string routines are often discussed as yet another programming foible. Programming language textbooks, e.g. [4], emphasize epilog activities of run-time stack management, but assume the stack remains unchanged. Operating systems permission structures assume control under a system administrator, not an attacker. Buffer overflow is sometimes cited to motivate software engineering defensive programming techniques.

Usually, the combustible mixture of data and control is glossed over. Divergent sub-disciplines of computer science and the consequent packaging into separate courses and accredited curricula partially account for missing this cross-cutting topic. However, external and public issues such as economic costs and product reputation sometimes force classroom attention. Various defenses against buffer overflows are available: more secure library routines [2], static analysis tools [5], a redesign of C called Cyclone [6], and intrusion detection [7]. Of course, the best defense is not to use vulnerable languages, but pragmatics, tradition, and past training suggest that the buffer overflow problem isn't going away anytime soon.

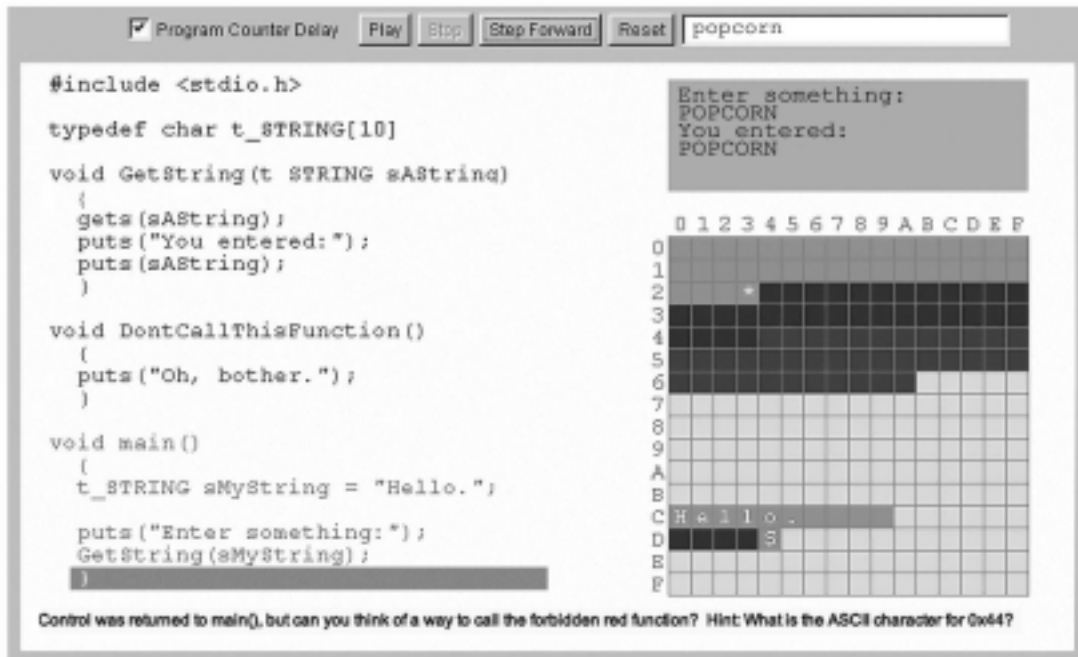


Figure 1. Screen capture of Java applet demo “Smasher”

3. Buffer Overflow Applet Demonstrations

Demonstrating buffer overflows as simulations of an abstract machine seemed like a natural starting point where student responsiveness to animation could be assessed and longer-term learning investigated. The case for using simulations in Information Security education is discussed at length in Saunders [8]. Our goal was to provide a forceful, intervention-style presentation rather than lengthy explanations. Students should respond with “That’s awful, my programming project team will never do that!”

Indeed, three computing course demonstrations and two noontime seminars to non-computing students evoked student excitement and questions not seen in typical lectures and labs. Feedback indicated a potentially high payoff from these 30-minute presentations. The prerequisite knowledge was approximately the level of beginning-programming students in C. Additional just-in-time explanations of the run-time environment of programs were needed, leading to a general tutorial description of stacks and run-time environments.

3.1 Using the Java Applets

The Java applets provide a visual and animated representation of the different concepts needed to understand buffer overflows. An abstract machine hides details that might hinder the student’s understanding, such

as the use of a specific memory architecture or assembly code.

A C program is shown on the left, with each subroutine displayed on-screen as a different color, keyed to the stack frames of the same color in abbreviated memory to the right. The code text for the entire program is arbitrarily placed at the memory address 0x00 and memory addresses 0x00 through 0xFF run left to right, top to bottom (Figure 1).

The user clicks on the “Step Forward” button to execute one line of C code or Play to watch the steps run automatically. The line of code that is being executed is highlighted. Program input and output occur in the box on the upper right.

A stack of activation records is arbitrarily placed at the memory address 0xC0 and grows upwards in memory. Each activation record is created on entry for that subroutine’s parameters, local variables, and a return address (denoted by a “\$” in the background color of the calling subroutine).

When the program requests input, for example by using `gets()`, the user can type the input in the text box at the top of the applet. If the buffer used to store the input is overflowed then anything that comes after it in memory is visually overwritten. The user of the applet plays the role of the attacker and tries to find an input string that will circumvent the imaginary security measure. Supporting text is a one sentence explanation, or hint, displayed on the bottom with every step through the C code.

For example, in the "stack smashing" applet, an attacker could enter an input that is 11 characters long with the 11th character being "D." The return address that points back into main() is overwritten. When the GetString() subroutine exits it will use the ASCII value for "D" (0x24) as the return address instead of what was there before, so the function DontCallThisFunction() is called.

3.2 Implementation Experience

Java's object-orientation facilitated the reuse of code and incremental development. The main applet class contains all of the code to draw the graphics and handle input and output. A separate class contains definitions for the data structures and functions to provide some C code and emulate the behavior of that particular C code.

Currently there are four demonstrations: the stack structure of activation records, a buffer overflow attack that overwrites data, the "stack smashing" shown in the figure, and a variation showing how the StackGuard [9] compiler works. Developing new demonstrations takes very little time because of the object-oriented approach. The biggest drawbacks of Java are that it isn't capable of producing graphics as detailed as something like Macromedia Flash and that applets aren't entirely portable from browser to browser.

Our animation approach differs from JAWAA [10] and Tango [11] in that we needed fine grain detail and control that fit nicely with the inheritance of abstract machine structures. The animation instructional technique is controversial, not well proven as effective, and still difficult despite the technology advances of Java [12]. Our experience is that demonstrations such as buffer overflow are hard to explain without some simulation, benefit greatly from abstraction, and gain momentum with a dramatic story such as attackers, defenders, and industry fiascos such as Code Red [14].

4. Contextual and Explanatory Content

The applet demonstrations provide stand-alone instruction for individual students, refresher instruction for professional programmers and testers, and demonstration instruments for instructors. However, there is far more contextual material worth integration beyond animation, as well as gaps in the software engineering literature treating buffer overflows. We therefore reverted to writing PPT, MS WORD, and PDF formats while experimenting with a highly effective, but difficult to use, Macromedia Authorware software tool [13]. Each of the following is available on the website:

- **Life Cycle Explanation**, a graphic, starting from vendor development techniques, into the escape of buffer overflows into products, their materialization as

security vulnerabilities, then responding alerts and patches, and the implied lack of feedback to academia motivating this material. This process-oriented presentation provides a framework for an instructor to motivate different stakeholder viewpoints, SWEBOK [15] issues, and driving questions, such as "why didn't testing method X catch buffer overflows?"

- **Code Red Case Study**, data and animated gif [14], describing consequences of the mid-2001 worm exploiting buffer overflows. Economics, infrastructure vulnerability, and professional ethics discussions are motivated.
- A **technical explanation** of buffer overflows for C programmers (and testers) digs into the nitty-gritty details and causes of buffer overflows, including examples. Demonstrations of **some intricacies of exception handling** are also provided. **Quizzes** challenge the experienced programmer.
- **Analyses of defense tools and techniques**, including a composite matrix of ratings of effectiveness of defenses against types of attacks. This content is a good summary or a large section of a technical software engineering course, also suitable as a refresher for practicing programmers and testers.
- A **non-technical explanation of buffer overflow**, using a mailroom metaphor, explaining computer memory and pointers from the ground up in story form, for journalists, industry analysts, or interested citizens.
- **Points to remember** and **checklists for code inspections** to improve software quality, in new software as well as legacy code.
- A **"scavenger hunt"** (odd facts about buffer overflows and vulnerable products) motivates economic issues and professional ethics.
- The **Authorware packaging** of this content provides a smooth interface with additional interactivity and navigation not available in PDF, HTML, or WORD documents.

5. Module Experience and Evaluation

Effective evaluation of educational processes can involve many strategies. For interactive learning systems, evaluations are usually conducted in three phases: an initial *needs analysis*, ongoing *formative evaluations* during development, and a *summative evaluation* at the conclusion of development. Although it is common for instructional products to be sold without any evaluation and revision process prior to distribution, studies have shown that even the simplest level of evaluative review with only one user can yield significant gains in learning effectiveness over products without such analysis. [15]

5.1 Evaluation strategies

The first step in writing the buffer overflow module was to perform a type of needs analysis to determine specific objectives and audience. For example, the construction of audience/objective and content/objective matrices helped to clarify audience needs and instructional content that were not anticipated by the subject matter expert and developer (Figure 3).

During development of the module, we performed a series of formative evaluations to “tweak” the instructional product as it was designed and written, with the goal of improving the module’s usability, effectiveness, and appeal. Both subject matter experts and potential users were involved in this process. Demonstrations were given to students in programming languages, software engineering, and cryptography courses, and to a group of McNair Scholars (a grad school prep program). Feedback on the content and delivery method was obtained by pre- and post-test comparisons, questionnaires, and group interviews with these students. (Figure 2.)

Pre-treatment quiz questions (written):

1. At a subroutines exit, how does control pass back to its caller using activation frames on the run-time stack?
2. What happens to nearby memory in a C program when a longer string is copied into the buffer, i.e. storage allocated to hold a string of a specific length?
3. A worm can make its own way as an independent program onto another computer. Describe how a worm program can invade another computer?
4. Describe a development practice that prevents worms from occurring?
5. Where would you recommend systems administrators and project managers look for security alerts and trends?

Post treatment focus group questions (interview):

1. Was the information presented new to you?
2. Was the material presented in a manner that was easy to understand?
3. What was most useful about the presentation?
4. What might be improved about the presentation?

Figure 2: Formative evaluation examples

Following development, evaluation focuses on the effectiveness of the module as a learning tool and on interface and usability issues. Evaluation of interface and usability is a relatively simple task, which we are performing with the assistance of targeted users and through user feedback via the project Web site. Targeted

users were asked to provide a review of the module, guided by a set of questions that examined the clarity and appropriateness of content, and navigational usability. In addition, the Web site feedback form includes an opportunity for comments on usability. Some users have reported various problems using the materials on their particular machines or lab set-ups.

The more important evaluation of the module’s learning effectiveness is a far more complex matter; an endeavor that cannot be accomplished with a simple “control-group” comparison, especially when few, if any, alternatives exist. Awareness, implementation, and accurate understanding of the module content are the most important issues in a summative evaluation of this module. Dissemination and observation are part of this process, but final results are pragmatically beyond the scope of the project.

Our initial observations of learning effectiveness involve students who have clearly benefited from the demonstrations and use of the module. Some of these examples have been self-reported from Web site feedback forms regarding how the materials are used and in what context. Students report that they are finding the material through search engines such as Google or being referred by other students. The modules are used for self-education, for extra credit in college courses, and to help other students. In cases of students on our campus who have been exposed to the module and demonstrations, it is too soon to know if students are making changes to how they approach the software engineering or programming process. However, we have observed students transferring the module concepts to programming contexts not anticipated by the project investigators.

5.2 What we’ve learned

The Importance of Defining Audience and Objectives. Although instructors generally accept the importance of defining audience and objectives, the value of this process became very evident as work began on additional modules. The different teaching styles of the project investigators produced varying levels of attention to this preliminary step. As we began work on other modules, it soon became clear that the less attention paid to defining the objectives and audience, the more difficult it was to progress to the next steps in development and to consider off-campus dissemination venues after completion. This was important, since an underlying goal of the grant was to produce modules appropriate for our undergrad curriculum, but also valuable to a wider audience. For one module in particular, the definition of audience and objectives was challenging, but after a few iterations proved essential to keeping the content within a doable framework.

The Challenge of Interactivation. “Interactivation” is the term we’ve coined to describe the process of enhancing static course material (for example, text) using methods that actively engage the learner in the content. Interactivation requires judicious and non-gratuitous development of interactions such as animations, illustrations, quizzes, Web searches, role-playing, puzzles, and problem-solving activities that support the content learning objectives. The pedagogy of interactivation is based in educational theory supporting the use of multiple cognitive paths in the learner to create a more accurate mental map of the content than would occur with the use of only one cognitive path. Interactivation of a topic can be a challenge; the process is subjective and complex, and is tailored to a specific set of objectives and audience. [19, 20, 21].

Interactivation requires educationally effective interactions, activities that engage the learner by: piquing interest, promoting discovery and reflective thinking, providing practice, creating a memorable experience and an accurate mental model of the topic, and providing an appropriate level of challenge. To be educationally effective, an interaction need not have all of these characteristics, although it should always create an accurate mental model. [22]

The general guidelines for interactivation are obvious to read, but are not necessarily so to implement. For example:

When a process is described, there is the potential to illustrate, animate, or simulate the algorithm to be more memorable and accurate than is possible with a text description alone. Interactive examples include the use of animated gifs, Flash, Authorware, or Java applets. To be interactive, these elements should be under the active control of the student. Use of Authorware was the primary delivery mechanism for the Buffer Overflow demonstration applets, PowerPoint slides, and exercises.

As is the case in classrooms and traditional practice, quizzes, matching exercises, crossword puzzles, etc. can be used to provide practice in learning terminology and definitions. An interactive learning environment has the advantage of varying presentation of the exercises (such as randomization) and to automate feedback and scoring. The demonstration applets and Authorware quizzes are examples of practice interactions in the Buffer Overflow Module.

When analysis and synthesis of related concepts is required, problem-solving exercises and Web searches can be used to promote discovery and encourage reflective thinking. Since the Web is a rich source of historical and current data, such as economics, case studies, current events (e.g. Code Red), etc, the potential of the Web as a interactive resource in designing learning exercises is enormous. Among the exercises provided with the Buffer

Overflow Module are a Web Scavenger Hunt and a list of suggested lab exercises.

The challenge of obtaining feedback. The Buffer Overflow Module was distributed via our project Web site, launched July 2002. Publication of the module was announced through relevant Internet news groups and invitations to targeted colleagues. The most detailed feedback obtained to date is through colleagues and recruited volunteers, including professors and a UNIX/C expert. As noted previously, we have also received feedback from forms available at the project Web site.

The value of student participation in the development process. Many students reinforce learning concepts by constructing concrete representations of knowledge, and being challenged to represent their knowledge accurately to others [18]. The quality of module content was enhanced by the expertise and enthusiasm of student content and applet authors, In return, these students benefited by the experience of being productive members of the development team. On the Buffer Overflow Module, for example, the demonstration applets and much of the content was written by an advanced student who understood software engineering principles, was deeply committed to security as a career and social importance, and who spoke the language of students (e.g., Star Trek references in the applets).

5.3 Conclusion and Next Steps

The buffer overflow topic lent itself well to the use of animations, but considerable information was needed to supplement the applet demonstrations. Observations of this multi-dimensional approach is thus-far positive, with students both on-campus and self-reported from Web feedback demonstrating the desired assimilation and transfer of the module content.

Other modules in this series. This project continues with development of three additional modules in the areas of Cryptography, Security Dimensions, and Personnel Security, capitalizing on lessons learned from the Buffer Overflow module. Evaluations and development of these modules are each unique and reflect the variety of teaching styles found among the investigators. These modules are in varying stages of development and the material is posted to our project Web site as the content is developed.

Acknowledgements

National Science Foundation Division of Undergraduate Education, NSF Award No. 0113627: *Increasing Security Expertise in Aviation-oriented Computing Education: A Modular Approach.* Website: <http://nsfsecurity.pr.erau.edu>.

References

[1] A. Ackerman, "Oracle Security Flaws Found", *Mercury News*, <http://www.siliconvalley.com/docs/news/svfront/secur122101.htm>

[2] G. McGraw and J. Viega, "Make Your Software Behave: Learning the Basics of Buffer Overflows", *IBM Developer Series*, <http://www.ibm.com/developerworks/library/overflows/>

[3] N. Frykholm, "Countermeasures Against Buffer Overflow Attacks", *RSA Tech Note*, http://www.rsasecurity.com/rsalabs/technotes/buffer/buffer_overflow.html

[4] M. Scott, *Programming Language Pragmatics*, Morgan-Kaufman.

[5] *ITS4 Software Security Tool*, Cigital Corp., <http://www.cigital.com/its4/>

[6] Morrisett et al, Cyclone, "A Safe Dialect of C", Cornell and ATT Research, <http://www.research.att.com/projects/cyclone/>

[7] Hoffmeyer, Forrest, Somayaji, "Intrusion Detection Using Sequences of System Calls", <http://www.cs.unm.edu/~steveah/jcs-accepted.pdf>

[8] J.H. Saunders, "Modeling the Silicon Curtain", *SANS Institute*, October 2001, <http://rr.sans.org/aware/curtain.php>

[9] "StackGuard: Protecting Systems from Stack Smashing Attacks", <http://www.immunix.com>

[10] S. Rodgers et al, JAWAA, "Java and Web-based Algorithm Animation". <http://www.cs.duke.edu/csed/jawaa/JAWAA.html>

[11] J. Stasko et al, "Algorithm Animation Research at GVU", <http://www.cc.gatech.edu/gvu/softviz/algoanim/algoanim.html>

[12] M.D. Byrne, R. Catrambone, and J. Stasko, "Do Algorithm Animations Aid Learning?", Georgia Institute of Technology, 1996, <ftp://ftp.cc.gatech.edu/pub/gvu/tech-reports/96-18.ps.Z>

[13] Macromedia Authorware and Flash, interactive software, <http://www.macromedia.com>

[14] Cooperative Association for Internet Data Analysis, "Code Red Visualization", <http://www.caida.org/tools/visualization/walrus/examples/codered/>

[15] IEEE Computer Society, Guide to the Software Engineering Body of Knowledge, <http://www.swebok.org>

[16] W. Dick and L. Carey, *The Systematic Design of Instruction*, 3rd Ed. Harper Collins Publishers, 1990.

[17] T.C. Reeves and J. G. Hedberg, *Evaluating Interactive Learning Systems*, University of Georgia, University of Wollongong (New South Wales), 1998.

[18] D. Jonassen, "Evaluating Constructivist Learning", *Educational Technology*, 36(9), 28-33. 1991.

[19] J. Hogle, *Considering Games as Cognitive Tools: In Search of Effective Edutainment*. ERIC no. ED425737, 1996. <http://twinspacefarm.com/pdfs/games.pdf>

[20] G. Kennedy, et al, "Integrating Computer Facilitated Learning Resources into PBL Curricula". *Interactive Multimedia Electronic Journal of Enhanced Learning*. 3(1), 2001, <http://imej.wfu.edu/articles/2001/1/02/index.asp>

[21] N. Sonwalkar, "Changing the Interface of Education with Revolutionary Learning Technologies", *Syllabus Magazine*, November 2001, <http://www.syllabus.com/article.asp?id=5663>

[22] L. Rieber, *Computers, graphics, and learning*. Brown & Benchmark, Madison, 1994.

Objectives	Audience	Professional programmer	Professional tester	CS332 advanced under grad CS students	CS225 beginning programming for CS and non-CS students	IT manager or worker	STG or other social science	Journalist
Understand about Buffer Overflow:								
Details of BO		✓	✓	✓				
Nature of BO					✓			
Caused by programmer error						✓	✓	✓
Occurs from oversized string input							✓	✓
Information about pointers:								
Reinforce practice of pointer checking while programming		✓						
Reinforce practice of white box testing for pointers			✓					
Demonstrate effects of pointer mismanagement				✓	✓			
Reduce long string vulnerability:								
Reinforce the need for testing long strings		✓						
Demonstrate the need for testing long strings in black box testing			✓					
Suggest need for testing long strings				✓				

Figure 3. Sample page from Buffer Overflow Module, Objective/Audience Matrix