Title: Driving Home the Buffer Overflow Problem:
A Training Module for Programmers and Managers

Authors: Jedidiah R. Crandall, Susan L. Gerhart, Jan G. Hogle

Affiliation:

Embry-Riddle Aeronautical University
Phone 928-777-3882, Fax 928-777-6945

Presenter: Susan Gerhart

Contact: Susan Gerhart

Address: gerharts@pr.erau.edu

Dr. Susan Gerhart
College of Engineering
Embry-Riddle Aeronautical University
3200 Willow Creek Rd.
Prescott AZ 86301

Abstract: Repeatedly, news headlines read: "Buffer overflow in vendor's product allows intruders to take over computer!" This widespread programming mistake is easy to make, exacerbated by the ubiquitous C language, and very simple to exploit. We describe a demonstration (a Java applet) appropriate for a traditional programming course to drive home key points: why buffer overflows occur, how overflows open the door to attackers, and why certain defense mechanisms should be used. The module is in its early stages of experimental use, with a formative evaluation to determine how well the module works and opportunities for its improvement.

Driving Home the Buffer Overflow Problem: A Training Module for Programmers and Managers

Jedidiah R. Crandall, Susan L. Gerhart, Jan G. Hogle
Embry-Riddle Aeronautical University, Prescott AZ

Abstract

Repeatedly, news headlines read: "Buffer overflow in vendor's product allows intruders to take over computer!" This widespread programming mistake is easy to make, exacerbated by the ubiquitous C language, and very simple to exploit. We describe a demonstration (a Java applet) appropriate for a traditional programming course to drive home key points: why buffer overflows occur, how overflows open the door to attackers, and why certain defense mechanisms should be used. The module is in its early stages of experimental use, with a formative evaluation to determine how well the module works and opportunities for its improvement.

1. Overview

We briefly describe the motivation for this module and background on the buffer overflow problem, and then describe the module's structure and operations, followed by our experience and evaluation so far, and plans for further development, assessment, and dissemination.

2. The Buffer Overflow Module

2.1 History of the Buffer Overflow Module

This module development and assessment is supported by a National Science Foundation Grant (http://nsfsecurity.pr.erau.edu) under the Capacity Building Track of the NSF Scholarships for Service program. The grant will develop at least five modules appropriate for undergraduate engineering and global intelligence students. Other planned modules are: personnel screening, Trojan horses, cryptography, and vulnerability testing. The security module team includes 3 faculty members in computer science, 1 in global intelligence, and a consultant in educational technology, plus several student assistants. The proposed modules may range in size from one lecture to a semester course, and vary in audience from undergraduate majors, through interested minors, to industrial short courses. We also seek to import modules to be adapted for our curricula.

Embry-Riddle Aeronautical University students are self-selected for future careers in aviation engineering, intelligence studies, airlines, and the military. Topics in security are highly relevant to the current engineering and intelligence programs, but a full-scale technical security track is only beginning to be considered. Therefore, the initial steps are targeted topics to (1) maximally increase the current security content of the curriculum, with the least perturbation of course sequencing, (2) assess interest from the students, and (3) increase faculty competence and involvement. In addition to fitting into existing curricula (1) the modules should be interactive using current computing technology, (2) we should apply standard methodology for designing and evaluating instructional modules, and (3) the end results should be disseminated to other training sites.

The Buffer Overflow Module was an obvious starting point, given the notoriety and persistence of the problem. It also fit well with our curricula, with its first programming courses in C. Starting from an in-depth web search that identified key papers, the undergraduate co-author developed the Java applet demonstration prototype. The initial target students were enrolled in CS 332, a “concepts of programming languages” course that surveys compilers and run-time environments, language features, and design issues. Following a short quiz to measure students’ knowledge of the relevant technology, two course periods of demonstration and discussion were then assessed by interviews, as described in the section on evaluation.

The current state of the module, early 2002, is the Java applet suite, with a short description at http://nsfsecurity.pr.erau.edu/bom.

2.2 Background on the buffer overflow problem

Among programming bugs notable to the public [News], the "buffer overflow" is vying with Y2K for top billing. While the details of the flaw and of exploits of its vulnerability may differ, the prominence is dramatic: 727 responses in the MITRE CVE http://cve.mitre.org, 487 at CERT http://www.cert.org, and 680 at InfoSysSec http://www.infosyssec.com.

A "buffer overflow" is said to occur when a pointer (as in C) goes out of range to access memory beyond the buffer. The major issue is what's accessible from where the buffer resides. 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. Getting the over-long string into the buffer requires additional knowledge of how the application works, e.g. when user input or system environment variables are called.

While web explanations are readily accessible [IBM,RSA], traditional textbooks do not directly address the problem. Inappropriate use of pointers and library string routines are often discussed, but the full, and very interesting, story of the buffer overflow chain of events is not covered. Pointer errors are treated at the level of one more programming bug. Likewise, programming language textbooks [Scott] describe stack management, emphasizing epilog activities of control flow return, but assume the stack remains unchanged. The combustible mixture of data and control is glossed over. In operating systems, permission structures are often thoroughly discussed but largely under the assumed control of a system administrator, not an attacker. And, in software engineering, the buffer overflow may be mentioned as a typical software security vulnerability, viewed primarily as a reason for more defensive programming without detailed description.

Why didn't testing catch these buffer overflows? It would seem that an adequately long string, not necessarily one meant to attack, would at least cause a program crash. Perhaps software testing, the step-child of all computing sub-disciplines, is another focal point for education and training to prevent buffer overflows.

Clearly, the perpetuation of buffer overflows into new code and their residual effects in legacy code, illustrate a systemic failure of both software technology and education. Perhaps the accountability issue is spread too widely through the different sub-disciplines of computer science -- programming languages, programming technique, software engineering, and operating systems -- and the consequent packaging into courses and accredited curricula. Cross-cutting issues are always difficult to handle.

Various defenses against buffer overflows are available: more secure library routines [IBM), static analysis tools [ITS], a redesign of C called Cyclone [Cyclone], and intrusion detection [IDS]. For a good survey of different countermeasures see RSA.

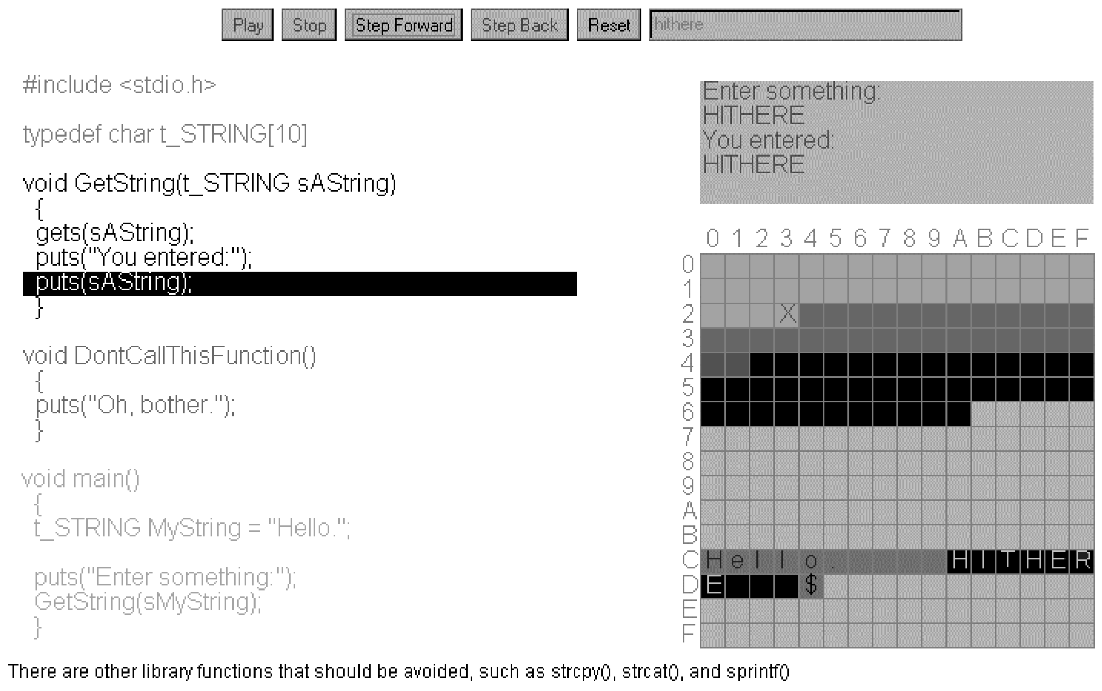
2.3 Description of the module

Our goal was not to demonstrate the variations of buffer overflows, nor to explain all the different modes of attack, but rather to get across the principles of the problem and to drive home the seriousness of buffer overflows. Our primary measure of understanding was that the student as a future programmer never makes a buffer overflow error and the student as future manager is able to take preventative actions and to control the effects of buffer overflow attacks.

The prerequisite knowledge necessary for using the module would be about that of a beginning-programming student in C. However, with a speaker presenting the module the audience might be able to understand the concepts without familiarity with the C language. Knowledge of the structure of activation records as stack frames is helpful but not necessary.

After viewing the presentation or working through the module on their own the student should understand the seriousness of the buffer overflow problem, the programming language, memory architecture, and operating system concepts that lead to its causes, and the consequences, as well as be aware of a number of countermeasures including StackGuard [SG], memory management patches, static checking, Cyclone, and intrusion detection. In many cases the information will be conveyed as supporting material with hyperlinks to existing literature rather than as part of the demonstration of the module.

The purpose of the Java applet is to provide a visual and animated representation of the different concepts needed to understand buffer overflows. An abstract machine was created for this that will hide details that might hinder the student's understanding, such as the use of a specific memory architecture or assembly code. The user of the applet can be a student trying to learn about buffer overflows or a presenter using the software as a demonstration, perhaps on an overhead projector.



```
#include <stdio.h>

typedef char t_STRING[10]

void GetString(t_STRING sAString)
 {
 gets(sAString);
 puts("You entered:");
 puts(sAString);
 }

void DontCallThisFunction()
 {
 puts("Oh, bother.");
 }

void main()
 {
 t_STRING MyString = "Hello.";

 puts("Enter something:");
 GetString(sMyString);
 }
```

```
Enter something:
HITHERE
You entered:
HITHERE
```

There are other library functions that should be avoided, such as strcpy(), strcat(), and sprintf()

A C program is shown on the left. In the color Java applet, each function is a different color. Each function has a corresponding code text in memory that is painted the same color. The code text for the entire program is arbitrarily placed at the memory address 0x00. In the figure the memory addresses 0x00 through 0xFF run left to right, top to bottom.

The user clicks on the "Step Forward" button to execute one line of C code. The line of code that is being executed is highlighted. The input and output of the program is shown 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 into a subroutine and contains that subroutine's parameters, local variables, and a return address to the place where the subroutine call was made from in the calling subroutine's code text. This return address is represented by a "$" whose background color corresponds to the color of the calling subroutine. It is assumed in aim of simplicity that all library functions, such as gets() and puts(), are compiled inline and therefore do not require subroutine calls. The term subroutine is used here to identify the separate units of code text in memory and not to identify different functions in the C code.

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. Only alphanumeric characters are accepted and all are converted to upper case. If the buffer used to store the input is overflowed then anything that comes after it in memory is 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. The only supporting text in the original version of the applet is a one sentence explanation, or hint, that is displayed on the bottom and changes with every step through the C code. The hint is used to point the user in the direction of exploiting the buffer overflow.

For example, in the applet above, 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. This particular lesson demonstrates "stack smashing."

The object-oriented structure of Java was utilized to facilitate 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. This class can have any number of children that, when instantiated within the main applet class, compile into an entirely separate "lesson." Currently there are four lessons: one to demonstrate the stack structure of activation records, another to demonstrate a buffer overflow attack that overwrites data, the "stack smashing" lesson shown above, and a variation of the "stack smashing" lesson to demonstrate how the StackGuard [SG] compiler works. Developing new lessons takes very little time because of the object-oriented approach.

Object orientation is not the only reason that Java is a suitable language for a learning module such as this. Java is well supported and widely used. There are many tutorials and code examples on the Internet. Because it is based on C++ it is easy to learn. The biggest drawbacks of Java are that it isn't capable of producing graphics as detailed as something like Flash and that applets aren't entirely portable from browser to browser.

3. Module Experience and Evaluation

Effective evaluation of educational processes can involve many strategies. For interactive learning systems, evaluations are usually conducted in two phases: ongoing formative evaluations during development and a summative evaluation at the conclusion of development. Studies conducted in the field of education have shown that thousands of instructional products sold in the United States do not go through any level of the formative evaluation and revision process prior to distribution. Yet other studies have shown that use of the formative evaluation process, even at its simplest level with only one user involved in the review, can yield significant gains in learning effectiveness over products without such analysis. [Dick and Carey]

Formative evaluations are conducted during development as an ongoing series of analyses. This process essentially “tweaks” the instructional product as it is designed and written, with the goal of improving the usability, effectiveness, and appeal of the interactive learning system. Both subject matter experts and potential users are engaged in the process to obtain valid and reliable feedback on the instructional module. Formative evaluation activities consist of an expert review, a user review, usability testing, and field-testing of prototype products. A summative evaluation is conducted to critically review the module after development is completed. The purpose of the summative evaluation is to determine overall educational effectiveness. This evaluation is often conducted with control and treatment groups to compare users of the newly developed instructional system with a control population exposed to a traditional or competitive system. Although the process sounds simple, it can be challenging to identify and standardize for comparison all of the significant variables affecting the test and control users of an educational system.

3.1 Evaluation method

The Buffer Overflow Module is in the initial stages of formative evaluation. The goals of the initial analyses were to (1) better understand the preliminary level of knowledge possessed by undergraduate computer science students of the buffer overflow problem, (2) obtain student feedback on the effectiveness of a java applet in presenting the material, and (3) obtain student feedback about possible modifications or additions for best effectiveness when the applet is presented without live interaction from the applet’s author.

This stage of the formative evaluation required several steps. First the professor developed a quiz covering buffer overflow concepts. The quiz was presented to a CS-332 class immediately before the applet presentation. Following the quiz, the student author presented the Java applet to the class. The material was displayed using an LCD projected computer screen, running the applet in a web browser. The author stepped through each example and explained the material to the class.

Two weeks following the Buffer Overflow presentation in class, students were asked to volunteer for brief group interviews to discuss the content and it’s implications. The interviews were not held immediately after the material was presented to reduce short term learning effects and to test retention of the new learned material. Three interviews were conducted. Two were “focus-groups” in which students discussed Buffer Overflow concepts and implications. A third interview with two addition students covered the same topics in a less formal session.

Questions asked of the students included:

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?

3.2 Evaluation results: What we've learned

As revealed by the quiz, preliminary knowledge possessed by our CS 332 students of the buffer overflow problem varied widely. Answers to the quiz showed that most students knew a buffer overflow involved an overflow of data or code that was overwritten, yet six of eight students were unable to describe details of how the problem worked or how it might be prevented. About half of the students had at least some idea that web sites existed for the purpose of disseminating security information, although most did not know exact URLs.

Students with some prior knowledge of the buffer overflow model were included in our evaluation. These students were helpful as "subject matter sophisticates who assisted in determining how less informed students would respond to the module presentation and content. Inclusion of sophisticated learners [for example, McNair scholars] will also be helpful in later stages of development in determining whether our module can bring these learners up to full understanding of the content. If our instructional module doesn't work with these learners, then it probably will not work for those with less knowledge of the subject. [DC]

The purpose of evaluation is to determine what the students have learned and that they have learned it correctly. However, students do not always learn the lesson as planned. New learning can conflict with prior knowledge and in some cases may interfere with correctly understood concepts. In the case of the Buffer Overflow Module, it was evident from the CS332 final exam results that some students in the course adjusted their understanding of stack management concepts incorrectly as a result of the Buffer Overflow Module presentation. Rather than being a setback, this discovery is an important piece of information to be used in refining the module’s content and presentation.

Student feedback on the effectiveness of a Java applet in delivering the buffer overflow material indicated a high interest in the animated presentation of the applet. Across all three groups interviewed, students expressed positive responses to the question of effectiveness. Few had seen the information in a book but most held the opinion that the animated presentation was probably superior to a book’s description when presenting information of this complexity. Specific mention of the visual stepping through the code was described by students as very good and even superior. The consistency of display across different examples was thought to be helpful.

Some students commented about possible modifications for best effectiveness when the applet is presented without live interaction from the applets author. Students mentioned the need for different levels of text to accompany the material if presented to audiences with varying backgrounds in C programming. An overview of the content prior to the examples was suggested, which would include a general description of C programming code, terminology (perhaps as a tool available as needed in a separate window throughout the module), references, and a description of important conceptual details to look for when viewing the examples. Students also suggested that the descriptive text accompanying the examples be expanded for better clarity.

Reliability of the interview responses appears to be high due to the consistency of feedback given in the three separate feedback sessions. Further reliability assessment will be conducted at later development stages. Validity of the analysis was not thoroughly examined at this early stage of the module’s development. The intent of the initial analysis was not to obtain a quantitative comparison of pre- and post-treatment results. Validity assessment is expected at later development stages.

3.3 Next Steps

The next steps in development of the Buffer Overflow Module will involve transfer of the live class presentation to a computer-based product. Our intent is to distribute the module through the Internet and on CD-ROM. The interface will be designed using authoring products such as Authorware and Flash (Macromedia). The computer-based product will build around the buffer overflow Java applet with the addition of supportive texts and detailed graphics that go beyond the capabilities of Java.

Feedback gained from the students in initial evaluations of the module will be incorporated into the authored interface, which we expect will undergo revisions as user feedback is gathered at each level of development. Evaluation of the module at the next stage will focus on assessment of the interface design and content incorporation. An interface assessment includes review of navigation, ease of use, manageable cognitive load, screen design, information presentation, media integration, aesthetics, and overall functionality. In addition, usability testing will assess the module for effectiveness of the ease of learning (a different issue from ease of use), minimizing user error, subjective user satisfaction, and maximizing user retention over time. [RH]

4. Conclusions

This work in progress addresses a serious, systemic problem: the programming technology and programmer mistakes that open the door for repeated attacks on critical infrastructure software across many vendors. The objective of the module is to provide an intense, lasting experience that will prevent future technical mistakes and provide a better understanding of the chain of security effects leading to current security fiascos. We have focused the demo applet on a distilled version of the problem intended for both individual and classroom use.

We are attempting to use available interactive technology (here Java applets) to maximize that student experience and break away from the lecture/book mode of learning. Our initial experience was well received with useful feedback on the students' changed perspectives as well as suggestions for improvement. Our module development will continue with additional demonstration experiences and their evaluations. The target date for dissemination of a well-documented, multiple-use package is July 1, 2002.

References

[**IBM**] G. McGraw and J. Viega, *Make your software behave: Learning the basics of buffer overflows*, IBM Developer Works Series http://www-106.ibm.com/developerworks/library/overflows/
See also *Preventing Buffer Overflows* at http://www-106.ibm.com/developerworks/library/buffer-defend.html

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

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

[**News**] A. Ackerman, *Microsoft, Oracle security flaws found*, Mercury News, http://www.siliconvalley.com/docs/news/svfront/secur122101.htm

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

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

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

[**SG**] *StackGuard: Protecting Systems From Stack Smashing Attacks*, http://www.immunix.com

[**MM**] *Macromedia Authorware and Flash*, http://www.macromedia.com

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

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

Supported by:

National Science Foundation Division of Undergraduate Education,
Federal Cyber Service: **Scholarship for Service (SFS)**
NSF Award No. 0113627
Increasing Security Expertise in Aviation-oriented Computing Education: A Modular Approach
Website: (http://nsfsecurity.pr.erau.edu)