A Strategy to Integrate Defensive Programming into the Undergraduate Computer Science Curriculum at UMBC

Doug Cress
cress1@umbc.edu

Brian Roberts
roberts1@umbc.edu

John Simmons
js5@umbc.edu

Computer Science Department
University of Maryland Baltimore County
May 14, 2003

Abstract:

As the utility and ubiquity of information technology (IT) have increased, the deployment of poorly-written, insecure code has exploded. Hundreds of vulnerabilities are revealed by security defenders and malicious attackers each year. This ever-widening gap between good, secure software development supposedly taught in schools and the actual bug-ridden code that is deployed will continue to plague our society and endanger our future.

To combat this growing plague, we investigate how the existing undergraduate computer science curriculum at UMBC might be improved by including defensive programming concepts within the four required software development classes. We utilize input and feedback from the course instructors to create materials for the undergraduate classes at the heart of the curriculum: CMSC 201 (Computer Science I for Majors), CMSC 202 (Computer Science II for Majors), CMSC 341 (Data Structures), and CMSC 345 (Software Engineering).

We believe that stronger programmers, who write secure code by habit instead of by force, will be produced by a curriculum that repeatedly stresses basic defensive programming techniques. Techniques such as avoiding dangerous function calls and post-development analysis are simple enough that even first year students can incorporate, resulting in a lifetime of more secure software development. Students and instructors need to learn that the responsibilities of computer security lie not only with cryptographers, network engineers, and operating systems developers, but with anyone who writes software.

Keywords:
Undergraduate Education, Computer Security Education, Secure Programming, Defensive Programming, Computer Science Curriculum

Table of Contents

1.0 Introduction……………………………………………..1
1.1 Motivation..1
2.0 Critical Review of Previous work...................................2
2.1 Current UMBC curriculum……………………...2
2.2 Federal Guidelines for Computer Security……..2
Education
2.3 The 2nd Annual Information Assurance…………3
Curriculum Development Workshop
2.4 "Computer Security and Impact on Computer…..3
Education"
2.5 "Application of Security to the Computer………3
Science Classroom"
2.6 "Panel on Integrating Security Concepts into…..4
Existing Computer Courses"
3.0 Preliminary Interviews...4
3.1 CMSC 201 5
3.2 CMSC 202 6
3.3 CMSC 341 7
3.4 CMSC 345 9
3.5 Department Head 9
4.0 Results...10
4.1 CMSC 201 10
4.2 CMSC 202 12
4.3 CMSC 341 13
4.4 CMSC 345 14
5.0 Open problems/Conclusions 15
5.1 Open problems 15
5.2 Conclusions……………………………………...16
6.1 Bibliography 17
6.2 Acknowledgements..17

Appendix A: Materials for CMSC 201
Appendix B: Materials for CMSC 202
Appendix C: Materials for CMSC 341
Appendix D: Materials for CMSC 345

1.0 Introduction

In the history of computing, security has always been an afterthought. Unless a particular application was designed for a security purpose, no thought was given to how a particular application could be exploited by malcontents or error-prone users. As a result of this, millions of hosts are compromised every year resulting in the loss of productivity, money, and privacy. Mercifully, computer science is a relatively new field with an older generation that will exit the playing field in the coming decade. There is still time to reform the computer science education arena before we are forced to endure a permanent institutional mindset which neglects security. Impressing young students now with the fact that they will be responsible for future computer security problems – and teaching them how to avoid such problems through defensive programming tactics – will undoubtedly present a brighter millennium where societies can comfortably and safely use information technology to enhance their lives and lighten their burdens.

1.1 Motivation

Unfortunately, in most undergraduate institutions, security is a topic covered only in elective classes. Thus it is common for students to graduate with degrees in computer science without ever receiving instruction on security issues. Through the personal experience of the researchers, it is evident that this problem is not at all unique to UMBC. If information systems are to be resistant to errors and malicious code, those who design the systems need to be made aware of security issues as they learn to program.

An important step towards this goal is to expose computer science students to defensive programming concepts early and often in their coursework. It is unnecessary to reconstruct an entire curriculum to this end. Instead, a set of learning modules could be included in existing courses to expose students to this material. A learning module is a set of slides, and possibly some exercises or project descriptions, that can be easily added to an existing course. Ideally, one module could be tailor-made for almost every course in the catalog since nearly every subject in computer science has some impact on the security of its users. The module would emphasize the security aspects most closely related to the course, perhaps exploring the underlying concepts of security principles learned in earlier courses.

To limit the scope of this project, we have chosen to evaluate and provide defensive programming modules for four undergraduate courses at UMBC: CMSC 201 (Computer Science I for majors), CMSC 202 (Computer Science II for majors), CMSC 341 (Data Structures), and CMSC 345 (Software Engineering). Teaching defensive programming principles and practices in these four courses will have the widest impact because all students graduating with a major or minor in computer science must complete them prior to graduation. Additionally these courses comprise the bulk of each student’s programming instruction. The skills they learn in these courses will effect their assignments in any other class or job situation for the rest of their lives.

We will proceed by first reviewing some of the previous work in the realm of defensive programming instruction. Then we recount the interviews and discussions we had with the instructors of the various UMBC courses. Followed by a list of our suggestions for improvement and the reaction of the instructors too our suggestions. Finally we provide some conclusions and open problems that still need to be resolved.

2.0 Critical Review of Previous Work

2.1 Current UMBC curriculum

We begin our review of previous work by examining the current core programming curriculum at UMBC. Upon entering the computer science degree program at UMBC, each student is required to complete a series of courses introducing them to computer programming and problem solving through software development. These three courses are CMSC 201 (Computer Science I for majors), CMSC202 (Computer Science II for majors), and CMSC 341 (Data Structures). Students majoring in Computer Science are also required to complete CMSC 345 (Software Engineering). These courses work together to teach the novice Computer Science student the fundamental concepts of programming in the C language, advanced and object-oriented programming in C++, and developing data structures. Each course is designed specifically for a certain level of programming expertise and each course builds upon the previous course material (requiring that course as a prerequisite) to give students a thorough understanding of programming concepts.

While these courses strive to teach proper coding standards, language functionality, and basic problem solving, they mention very little about secure programming practices. The general approach is to instruct students on which behaviors or functionalities of the language to avoid, without giving a thorough explanation of why these behaviors are dangerous. We feel that in order for students to fully understand the implications of security (and the lack thereof) it is important to focus on these topics more thoroughly and present a more technical and in-depth examination of the concepts (within the scope of expectations for the respective courses).

2.2 Federal Guidelines for Computer Security Education

Since 1999, the National Security Agency has designated 36 universities as Centers of Academic Excellence in Information Assurance. In accordance with this designation these universities must adhere to a list of 10 qualifications including an overall curriculum mapped to various National Security Telecommunications and Information Systems Security Standards (NSTISSI). The foremost of which, number 4011, is the "National Training Standard for Information Systems Security (INFOSEC) Professionals," dated June 20, 1994. This standard is meant to be viewed as a curriculum to teach students how to be INFOSEC professionals. Sadly, this standard is not sufficient substitute for our work. It focuses primarily on policy and general Automated Information Systems (AIS) management and planning. Though these are important components for a graduate level course due to their more theoretical nature, this level of abstraction does not contain sufficient detail for teaching undergraduate students how to write code in a secure manner. Since software is a key component of any AIS, we believe that secure software development should be addressed at every step on the path towards a bachelor's degree in computer science.

2.3 The 2nd Annual Information Assurance Curriculum Development Workshop

In June of 2002, Purdue University sponsored the second annual Information Assurance Curriculum Development Workshop. Melissa Dark and Jim Davis chronicled the experience in a paper called: "Report on Information Assurance Curriculum Development." The workshop split up the participants into two groups: one focused on graduate level studies and one on undergraduate studies. The groups sought to classify the knowledge that students needed to attain into three domains: Declarative (memorization of facts & figures), Application (ability to use declarative knowledge previously acquired) and Synthesis (to create new solutions from existing knowledge). Out of the fourteen broad subject headings, only one was dedicated to secure software engineering practices. The four subjects listed (security of large software systems, programming language issues, software engineering techniques, and security issues) were good broad observations of the types of skills that the students should learn. However, detail was again lacking. A description of exactly what portions of an IA curriculum a student should be responsible for at a particular stage in his or her education was missing. A set of specific pitfalls and ways to avoid them in software engineering was not included. We hope that our work can start with these areas and provide specific solutions relative to the UMBC student, but generic enough in scope to allow other universities to benefit from our ideas.

2.4 "Computer Security and Impact on Computer Education"

Yang [YA01] examines the need for security to be taught in undergraduate curriculums. Many reasons why this is needed are given, and several outside sources of material are provided which may be helpful to those designing security lessons. Integration of security topics into an existing curriculum is examined in detail, with suggestions for each course, though no details are provided. Some time is also spent discussing the obstacles and issues facing anyone trying to enact real changes to a university's curriculum.

2.5 "Application of Security to the Computer Science Classroom"

Vaughn [VA96] covers similar topics in a much more abbreviated fashion. Specific recommendations are made for course content, but not as many courses are considered as in Yang's paper [YA01]. The courses treated are: operating systems, database, software engineering, computer networking, and artificial intelligence. It is interesting that for all its concern with security topics in advanced courses, this paper does not touch on including security instruction in introductory programming courses.

2.6 "Panel on Integrating Security Concepts into Existing Computer Courses"

Mullins [MU02] writes a brief introduction to a panel discussion that was held at the SIGCE '02 conference concerning the integration of security material into existing Computer Science courses. This matches almost perfectly with our project. Unfortunately, this document only contains position statements from each of the panel members, and a transcript of the discussion does not seem to be available.

3.0 Preliminary Interviews

In order to develop material which will be usable in the courses at UMBC, we felt that the most effective tactic would be to solicit input from the instructors who teach those courses. Before developing our material, we first conducted a round of preliminary interviews with the professors to give our project more direction. In this phase, we sought to determine an appropriate format and quantity of content for each class. Since it has been many years since any of the investigators were enrolled in equivalent classes, we also sought to verify that the proposed technical content of our new materials would be appropriate for students in these classes. Finally, we investigated the possibilities for including some of our material in projects and assessments in order to reinforce student learning. Additionally, we asked if the instructors had different opinions about which classes should be infused with secure programming material.

The following is a list of the core concepts we want to include in our four classes:

- The potential harm of programming errors and poorly secured code
    - Physical damage that has resulted due to unreliable code
    - Widespread malicious cyber-events against commercial companies, government agencies, and educational institutions
    - Financial impact of stolen credit cards or more serious bank account fraud
- Common insecure practices that might be encountered at a student's current instruction level
    - I/O data not validated
        - Can result in unexpected behavior in programs and invalid results
    - Segmentation faults due to accessing memory outside the bounds of an array
    - Data printed from the wrong memory addresses
        - Can potentially access and send sensitive information
    - Use of insecure functions such as strcpy() and gets()
    - Mishandled exceptions
    - Object Oriented programming issues
        - Class Destructors that don't clear out sensitive data before the memory is deallocated
        - Lack of properly designed access control mechanisms utilizing public/private/protected/package modifiers

        - Read-only accessor functions that return pointers instead of copies of the data referenced
- Poorly commented and non-standardized code and how it can contribute to security problems.
- Buffer overflows
    - What are they?
    - What is a buffer overflow attack?
    - How can buffer overflows/attacks be avoided?

3.1 CMSC 201

Sue Bogar and Dawn Block were the instructors interviewed for CMSC 201. This course is the introductory Computer Science course for majors, and focuses on teaching them how to code in the C language. Most of the students in this course are freshmen with very little formal programming experience or computer science skills in general. According to the instructors, many of the concepts of defensive programming might elude the students' limited understanding. However, the instructors believed there were several foundations that could be laid in 201 and embellished during later courses.

The instructors maintained that there are not many places where one can specifically talk about security and give detailed examples because of the students' limited experience. However, they did feel that there were plenty of opportunities to include "pointers" – a few sentences of material – throughout both the lecture and discussion notes. The instructors also described opportunities to make sure students utilized defensive programming practices in the five projects they are assigned during the semester. Students could be encouraged to pay attention to these techniques by requiring them to discuss any security implications of their project in their design documentation. Finally, questions could be added to the exams addressing some basic security issues as they relate to I/O validation and proper error handling.

Both instructors were a bit hesitant to alter the curriculum, primarily because they felt that the existing course material is already overwhelming for the students. Explaining basic security issues like buffer-overflows would be difficult due to the students' lack of experience with concepts such as stacks, heaps, memory management, and advanced programming practices. Both instructors seemed to think that advanced security concepts belonged in some course, but not in 201.

One of the first steps that the instructors thought would be beneficial is to incorporate warnings into the lecture whenever possibly dangerous behaviors were presented. The key concept of 201 is to teach students *how* to program effectively, using C as the medium for instruction. To this end, there is already great stress placed on I/O validation. Further emphasis could be achieved with security case studies to really drive the point home. Unfortunately neither of the instructors was very familiar with security issues beyond the basic dangers inherent to the language. Yet both instructors thought that it would be a good idea to discuss potentially dangerous functions such as *gets()*.

Currently, this is mentioned in lectures, and then the TA's are asked to go into greater depth during the discussion section.

One of the greatest obstacles to modifying this course is that while lecture notes are written each semester by the instructors, TA's take turns writing the notes for the discussion sections. This benefits the TA's by providing them the opportunity to gain experience in preparing teaching material. Unfortunately, this also results in notes which change every semester, and do not follow a standardized format. The discussion section notes are merely required to discuss lecture topics in greater depth. Thus, each TA tends to incorporate his or her particular strengths and interests into the discussion notes. While the notes are peer reviewed, and ultimately accepted or rejected by the instructors based on the validity of the content, there is no requirement for security currently stressed. It would be easy to examine three different versions of the same discussion from three different semesters and pick out similar concepts, but those concepts would be addressed differently depending on the TA's background and the examples used. This makes standardization difficult because the pool of TA's is constantly changing. Even if instructors are convinced to incorporate defensive programming practices into their lectures, each semester's TA's would have to be reminded which security concepts to stress.

3.2 CMSC 202

Mr. Raouf and Mr. Frey were the instructors interviewed for CMSC 202. Currently, Mr. Frey is the course coordinator. The stated goals for the class are to teach students general problem solving techniques, recursion, asymptotic algorithm analysis, basic data structures (linked lists, stacks, queues, binary search trees), abstract data types, memory allocation, functional parameters, basic sorting algorithms, object-oriented programming and C++ in general, and good coding practices. Learning of the course content is reinforced through five programming projects and three exams.

Certainly defensive programming practices fall under the purview of "good coding practices", so this course indeed seems to be an appropriate place to include our new material. After talking with the instructors, it was apparent that small amounts of new material could be easily added to the lectures. It was recommended that pointing out the relevant security pitfalls as new topics are covered would be much more effective than saving all security discussion until a lecture at the end of the class. Also, the course is already very full of information, and it would be difficult to add a single block of new material anywhere in the course. Thus making modifications on the order of a single slide or less to the relevant topics will make our results most easily adopted by the instructors, and should also be effective for the students.

Including defensive programming in the projects and exams would be more difficult. The exams are already loaded enough with the existing content that including more security-related questions would likely be hard to accomplish while still making sure that students are tested thoroughly on the existing material. The projects follow a similar pattern, where adding another grading component would put too much strain on both students and graders. For example, the projects already list user input validation as a grading component, but in practice it is not strictly enforced. Apparently, if more emphasis was put on it, students would spend too much time working on that aspect, and not enough time learning the other larger concepts of the project. Also, since input validation is stressed so much in 201, it doesn't need quite so much attention in this class. It would, however, be both easy for the instructors and interesting for the students to have a project whose scenario raised security implications (i.e. working for a credit card or medical company). The instructors suggested that a project regarding check digits for credit card numbers might suggest a good security theme. Unfortunately, the later projects – the ones scheduled after our new content – are closely tied to the lecture material. In teaching concepts like inheritance and polymorphism, the context of geometric shapes is used. This context is extended and reinforced by the projects, and changing one of the later project scenarios would destroy the beneficial qualities of that relationship between lecture and project.

We also gained some insight into what content we should include for the class. Students in this class should be able to understand concepts regarding memory management and clearing sensitive data. For example, it might be possible to examine what occurs in a malloc request, but the instructors would not be able to go into too much depth with operating system concepts, as the students have not taken that course yet. A demonstration of how random old data can be picked up in freshly allocated memory would be appropriate. Students would probably not be able to really understand how a lack of bounds-checking would lead to a stack-smashing exploit, but this is an appropriate place to demonstrate how failing to check bounds can lead to neighboring data accidentally (or intentionally!) being overwritten. An effective demonstration of how mishandling pointers can lead to client programs with access to private data members would be good. A recurring theme in the feedback was that our information should be concise, and reinforced with effective and dramatic demonstrations to capture students' attention.

3.3 CMSC 341

Mr. Frey, Mr. Edleman, Dr. Peng, and Dr. Oates are the instructors interviewed for CMSC 341. Currently, Dr. Oates is serving as the course coordinator. This course has approximately 29 class meetings in which teachers cover topics relating to various data structures and their implementation. The class focuses on Abstract Data Types (ADT) and how to solve problems using them. ADT's that describe how to manipulate stacks, queues, trees, heaps, hashes, and graphs are covered. Student learning is reinforced through 5 programming projects assigned throughout the semester as well as three exams.

In an attempt to be as practical as possible, our suggestions for incorporating defensive programming tactics for 341 revolved primarily around the class projects. Since this is where the rubber meets the road for the students, we felt that impacting them here would help reinforce the concepts better then merely lecturing to them. Of course the students need to learn how to incorporate defensive programming into their projects before they start, so we felt that some lecture instruction would be needed as well. In order to evaluate what the students had learned we also inquired about the possibility of incorporating a security related test question on one or more of the exams.

The first instructor interviewed felt initially that adding secure programming concepts might be simply too much work for the students to handle. He pointed out that most students consider 341 to be a challenging class and that adding additional security requirements to their programs and projects might be overwhelming. After further discussion, he felt that perhaps the best way to make the students more aware of security and defensive programming practices was to incorporate security related issues into the project descriptions in order to attract the students and make the projects more interesting. This has the benefit of not taking up lecture time and yet still exposes them to the necessity of defensive programming concepts. Each of the projects assigned in 341 has two or three related questions that must be answered by the students. These questions are designed to make the students think about what they have done and why they solved the problem the way they did. The instructor thought that a question about something relating to security or defensive programming would fit nicely here. Additionally he felt that the best places to include defensive programming tactics was early in the semester, during the lectures on general programming techniques and debugging. Then as a reinforcement tactic, he thought that a review of some of the security concerns that the students should be aware of could be included in a lecture at the end of the semester. Finally, the instructor was adamant about not including security related questions on any exam.

The second instructor interviewed did not feel that secure programming techniques belonged in the 341 curriculum at all. However after we had explained to him our ideas about how security should be woven throughout the CS curriculum he relented a little. After laying out all of our suggested improvements and ideas about where best to include defensive programming tactics, the instructor agreed that they were all good ideas. Unfortunately he was not very suggestive and did not provide any additional input as to how to incorporate security into the 341 curriculum.

The third instructor interviewed was a wealth of suggestions and input. His initial concerns were very similar to the first interviewee that the students taking 341 are fairly overwhelmed by the class and its subject material as it is. He pointed out that in order to impact the students, security and defensive programming needed to be presented as more then mere meta-information. For example, students are told to include comments in their code, but the comments do not seem to be very important to them and therefore are not given the serious consideration they deserve. He pointed out that students at this level are best taught by specific example and that grand esoteric concepts will not be absorbed very well by the students. Similar to the first instructor, this instructor felt that perhaps the best place to include lecture notes on defensive programming would be at the front-end of the class, during the lectures on good software development. He reinforced the idea that providing avenues for the instructors to teach their students should be easy for them to include and not generate too much extra work for them. Perhaps this instructor’s most practical advice related to the development of object-oriented class destructors and ensuring that the memory areas controlled by the destructor were wiped clean before they were released. He also felt that the best way to teach defensive programming to students was to suggest security themes and introduce defensive concepts to them at the lower classes and then to provide a 400 level course that dealt specifically with the security topic. Lastly the instructor thought that the most difficult aspect of incorporating defensive programming techniques was not teaching the students, but convincing the instructors that it should be included in the first place. He recommended talking with the department head to discover his opinions on the matter.

The fourth instructor interviewed, like all of the other 341 instructors, felt that 341 was perhaps not the best possible fit for defensive programming concepts. However, he did believe that incorporating defensive programming in the final lecture of the semester would provide some benefit as it would give the students something to think about before they arrived in 345. CMSC 345 is the software development class that all undergraduates are required to take. This instructor believed that 345 was a much better place for security to be taught and that warming the students up to security in 341 would be useful. He stressed that any slides offered to the instructors should be concise and easy for them to understand. Finally, the instructor offered the opinion that students taking 202 would not be interested in security concepts at all.

3.4 CMSC 345

Acting on advice from our other interviews, we decided that CMSC 345 would also be a good place to include some new secure programming material. This course is required for all Computer Science majors, but not minors. Still, it is very attractive to us since it is the first class where students are working on a large group project. The students in this class are split up into groups and then assigned a customer for whom they must develop a software package. The package requires the interaction of several components including internet related applications and local end users. This allows us to emphasize some secure programming concepts which have a more profound effect on larger projects.

Ms. Mitchell is the instructor we interviewed for CMSC 345. In talking with her, it was apparent that there is plenty of room for new material in the middle of the course, as students work on their semester-long project. An entire lecture on secure programming would be easy to include in this course. While we would need to be a little careful with technical content since students have not yet taken the operating systems class, this would be a good place to put our “capstone” secure programming material, reviewing and reinforcing the concepts learned in our previous courses.

3.5 Department Head

Following the suggestion of several of the instructors, we also interviewed Dr. Pinkston, the UMBC Computer Science department head. Dr. Pinkston thought the basic idea of teaching security early and often was a great premise. He provided several excellent suggestions including the focus on defensive programming versus secure programming as a more accurate description of what we were advocating. Similar to the other instructors he felt that adding defensive programming to 201 and 202 might not be the best course of action but that incorporating such topics into 345 and possibly 313 was a better idea. He suggested that if we were to insist upon including defensive programming in the earlier classes, then the best place to fit in the concepts would be within the 'best programming practices' section of the class. Dr. Pinkston pointed out that CMSC 313 (Computer Organization & Assembly Language Programming) might be a great place to add security concepts because of its focus on instruction set architecture and its relevance to buffer overflows. He felt that CMSC 345 (Software Design and Development) was an excellent capstone class for defensive programming. Dr. Pinkston felt that focusing on the security implications of requiring several programs to interact safely and the dangers inherent when one component fails would be especially germane to this class. Dr. Pinkston also mentioned that merely trying to scare the students into security would have no more effect then crying wolf. He recommended taking specific examples from the kinds of programs the students had written earlier in their academic career and showing them where insecure mistakes could have been included. Illustrating the possible ramifications of such errors, would be a much more effective training mechanism then just trying to frighten the students. Lastly because Dr. Pinkston was the department head, we asked him how we could go about convincing the department as a whole to adopt our suggestions into the curriculum. He pointed out that the department had already made commitments to the Federal Government to add additional security concepts to the UMBC CS curriculum. Additionally he mentioned that if a majority of the department felt that security issues should be included in the curriculum, then any individual resistant instructors would probably capitulate.

4.0 Results

For each of the four courses, we have produced new lecture material (see the appendices) as well as suggestions for improving projects and other assignments. Where existing lectures have been modified, we will make available a copy of the complete original lecture notes or slides with our modifications included. This should make it easy for instructors to make use of our suggestions. After completing a draft of our changes, we revisited the course instructors to verify that we had produced useable results. The outcome of those interviews will also be mentioned with each course's suggested new materials.

4.1 CMSC 201

We recommend the following changes for the lectures within the CMSC 201 curriculum. Overall, this course does an excellent job preparing students for their career in computer science. Data validation and structured programming are stressed, and basic problem solving is taught through rigorous projects and examinations. Addition of too much material regarding defensive programming practices at this level could overload students, so the volume of information should be minimized. Any concepts that are introduced should be tied to the current curriculum, and should stress the fact that secure code is also easier to manage and debug. We provide the following recommendations for lecture additions, with the understanding that while the order of the lectures sometimes changes slightly from semester to semester, the basic concepts covered in each lecture remains the same.

Lecture 2: Control Structures, booleans, cautions
- Introduction to data validation should occur in this lecture.

Lecture 4: Tracing Function Calls
- Mention the fact that the C compiler *will* let you do things that are technically insecure and dangerous. The fact that it *lets* you write bad or insecure code does not justify such practices.

Lecture 5: Purposes of Functions and Separate Compilation
- Stress the importance of testing and how modular programming can lead to less testing and more reliable, secure code through code reuse.
- Discuss the dangers of ignoring compiler warnings.

Lecture 6: Top-Down Design / Designing a Project
- Discuss the importance of keeping security in mind when designing programs.
- Mention libraries that might be helpful.

Lecture 7: (a link to the first project)
- This project should grade students' use of data validation techniques.
- The design for this project should also include a brief discussion of security implications.

Lecture 8: Arrays
- Discuss the dangers of referencing data outside the bounds of an array.
- Discuss the basic concept of a buffer overflow.
- Discuss segmentation faults.
- In the second project, students should be graded for avoidance of these errors. The design document should discuss the implications of using arrays and the possible dangers.

Lecture 9: Passing Arrays, Sorting and Searching
- Carry the discussions from lecture 8 over into this lecture, since they are basically dealing with the same concepts.

Lecture 11: Structures and Arrays of structs
- Discuss the fact that structures do not allow for any protection of data. Let the students know that C++ will introduce classes, which can remedy this problem.

Lecture 12: Structures and Arrays of structs
- Carry over the discussions from the last lecture. Ask students why it could be beneficial to protect data stored within a structure-like data type.

Lecture 13: Pointers
- Introduce the concept of pointers, discuss initializing pointers to NULL.

Lecture 14: Pointers Revisited

- Show how pointers are just like arrays and therefore subject to the same sorts of vulnerabilities.
- Make sure students discuss any security implications that pointers might cause in their project.

Lecture 15: Pointer Applications

- Discuss how malloc() works, and how any memory allocated by malloc contains whatever data previously existed at that memory address. Discuss the implications of this.

Lecture 16: Pointers to Pointers, Command-Line Arguments

- Stress the importance of data validation when dealing with command line arguments.
- Explain that pointers to pointers and arrays of pointers can have the same security problems as traditional arrays and pointers.

Lecture 17: Characters & Strings

- Discuss some of the potential insecurities of the string.h library.

Lecture 18: Streams, I/O and stdio.h

- Discuss the dangers of gets() and other insecure functions dealing with strings and characters.
- Make sure students avoid using insecure functions in their projects.

Lecture 20: Pointers to Structs and Self-Referencing Structs

- This lecture introduces self referencing structures in preparation for teaching linked lists. Be sure to stress the importance of initializing pointers to NULL after declaration.

Lecture 25: Stacks and Queues

- Introduce the concept of a heap and a stack and let students know that they will learn more about them in future classes.

Lecture 26: Memory Management

- Discuss the dangers of printing un-initialized memory space.

The instructors teaching 201 thought that adding too much material to the curriculum could overload the students. They also felt that much of the material required to understand defensive programming concepts was beyond the understanding of students at this level. However, they were supportive of adding a few references to security issues.

4.2 CMSC 202

In this course, we have inserted material into three lectures throughout the semester. The lectures already mentioned some of the topics we were concerned with, but we felt that the security implications were not emphasized enough. Our new material will add perhaps half an hour of additional content to the semester's lectures, and should not impact any single lecture tremendously.

A list of the lecture modifications in this course follows:

- Lecture 2 (Standards and Practices)

    - In "Why Bother with Standards" section (whystds.shtml), added bold text.
      Listed motivations for students to use secure programming practices
    - In "Coding Practices" section (CodingPractices.html), added bold text.
      Extended the precondition example to include a section about bounds checking and how it relates to stating preconditions
- Lecture 8/9: Pointers and Dynamic Memory
    - Added slide 10
      Explained that newly allocated data is not blank
    - Added slide 23-24
      Explained how deleted data is not cleared, how that relates to slide 10, and security implications of that
- Lecture 10: Copy/Assignment
    - Added slide 7-8
      Extended discussion of default copy constructors and how they can interact with pointers to create security problems
      Used this to illustrate how being imprudent with pointers can create security problems

We decided against modifying the project assignments for this class. As mentioned before, the later projects share a sort of symbiotic relationship with the lecture material which we do not want to disrupt. Earlier projects are assigned before most of our new material has been covered, so at best the project changes would be a review of material from CMSC 201. We feel that this is best left in the earlier course.

The instructors were very supportive of these changes, and felt that they could be easily included into the course. This course is already in the process of being re-evaluated and modified by the coordinator, so now is a good time to introduce these changes. Mr. Frey suggested that if some content regarding secure object design would be very beneficial, but we were unable to find good material of this sort to include.

4.3 CMSC 341

CMSC 341 is the data structures class at UMBC. As such it was important both to the researchers and the instructors that we not change the focus of the class from its appointed purpose. In order to accomplish this we first needed to understand what the course syllabus set forth to teach and what the course text book had to say in relation to security and defensive programming. Sadly, neither source mentioned anything about security whatsoever. From our preliminary instructor interviews, it was apparent that a few slides provided at the beginning of the semester and some reinforcement in one of the projects would be a good way to teach the students defensive programming tactics.

We developed seven slides that could be integrated into the current lecture series on basic programming issues. These slides covered topics first presented in 201 & 202 and included some new material as well:
- Good Comments
- I/O Validation

- Object Defense
- Dangerous Functions
- Post Processing/Static Analysis

We also revised Project 3, emphasizing some of the security and defensive programming tactics learned earlier in the semester. The project was basically to make a Binary Search Tree (BST) that held bank customers' account numbers and credit card numbers. This project does not change the core BST project currently used in the class, but does expose students to considering security issues like static analysis, I/O validation, and proper use of encryption to protect sensitive data.

After developing our slides and the programming project proposal, we met with the various instructors for 341 again. In general the instructors were receptive to our additions. Some were quite interested in incorporating our ideas into their curriculum immediately. The instructors who currently lead the 341 instruction team, felt that the new additions were quite valuable and would do well to augment the students understanding of defensive programming. Unfortunately they felt that our proposed project may be a little more difficult than the students could handle. Still, the questions and tone of the project description were exactly the type of improvements they were expecting. One of the instructors, in particular, enthusiastically requested copies of our materials for use in his class next semester.

4.4 CMSC 345

In this course, we have suggested some modifications to the project description, and created an entire lecture regarding secure programming. Also, the textbook currently used for this course does include some material on secure programming. We have tried to incorporate this material into our lecture, and several of our slides were produced directly from material in the book.

The lecture we created comes in two parts. First, we examine secure programming techniques. Concepts learned in previous classes are reviewed and extended, and special attention is paid to the effects of a large group project. Second, we present the anatomy of a stack-smashing attack, and ways such attacks can be avoided. This material has been modified from William Byrd's presentation on the same subject in CMSC 791. We have had to strip down the material both for time and technical content. However, we feel that this subject is important enough that students should be exposed to it at some point in the curriculum.

We have suggested some small modifications to the project description concerning secure programming. Security concepts have been added to the project description, and use of static analysis tools has been suggested. Students have also been encouraged to add security to their project through freely available software modules found online.

Ms. Mitchell felt that the first part of the lecture fit very well with the course. She suggested that the second part of the lecture be removed to a course such as CMSC 445. We agree with this, since the later class would allow us to cover that topic more in-depth. However, CMSC 445 is not offered frequently enough, so we will include our material to be used as a basis for future work. Ms. Mitchell was supportive of using static analysis tools for the project, as it is a topic included in the textbook which she is also trying to include more in the lectures. She felt that downloading freeware components and making security a required part of the project would be best moved to CMSC 445. We agree that moving those elements to a more advanced class would be appropriate, and will still include them here as a basis for future work.

5.0 Open Problems / Conclusions

5.1 Open Problems

First and foremost, the short timeframe of this project has not allowed us to evaluate the real effectiveness of our modifications. If these changes can be enacted, their effectiveness certainly needs to be assessed. We have several suggestions to this end.

Our first testing suggestion would be to conduct surveys by randomly sampling students at various levels within the CS curriculum to ascertain their current experience and knowledge of security and defensive programming issues. This would form a control for the experiment. Once our changes are enacted, a random sampling of students would be surveyed each semester to see how much they had learned in the different classes. By asking the same questions of all the students we would expect to see most of the questions left unanswered by the students attending earlier/lower level classes and more answered by students in the latter classes. This type of testing methodology might take several years to complete, and faces the usual problems of obtaining good survey responses, but could provide a very accurate picture of how effective our methods had been. This evaluation solution could be carried out with minimal impact on the instructors themselves.

A second evaluation technique would be for later classes to assume that earlier classes had covered particular issues. These later classes would be able to begin instruction at a particular stage and be able to grade the students' comprehension to this point in the CS curriculum on defensive programming and security awareness. This evaluation solution would require much more work from the department and the various class instructors. Though this method might clearly show the level of comprehension the students had attained, but the amount of work required could be prohibitive.

A third evaluation technique would be to offer a contest of some sort to upper level students (juniors and above). The goal of the contest would be to see if the participants could: given a real-world project specification, develop a software solution for it, and then submit it to the contest judges for a security audit. The prize would have to be substantial enough to get the students interested in participating in the contest. Perhaps an outside vendor would be willing to sponsor such prizes. This would have the added benefit of involving the students with sponsors and exposing them to possible job opportunities after graduation. This type of contest would not be preceded by any particular class teaching security. Instead the students would be expected to have learned enough throughout their various classes at UMBC to enable them to write secure code. The shortfall of such an evaluation technique would be the difficulty in proving the impact of our educational ideas on the students work, versus their innate talent and propensity for security. However if such a contest were offered to several universities in the area then perhaps the curriculum additions we are proposing could be a decisive factor in a UMBC team of students winning the competition.

Another open problem in this area is to extend the modifications to every course in the curriculum. Ideally, security modules and modifications could be developed for each course, emphasizing the secure programming concepts and general security issues associated with the material in that course.

Finally, an entire secure programming course could be created. It may or may not be difficult to find a semester's worth of material on secure programming, but the effort should be made. Our modifications in this project have been small, and aimed at impacting every student somewhat. This approach has limited the depth of technical content we were able to include. It would be wonderful if a 400-level elective class could be created for students interested in secure programming who already have the background provided by their earlier courses.

5.2 Conclusions

The prevalence of insecure information technology resources has been widely observed over the last decade. As we become increasingly reliant on these technologies, we need to ensure that our future reliance can be grounded in the knowledge that the software which drives these technologies was created by people who strive to ensure that it is safe and incorruptible. Teaching defensive programming tactics to each student who graduates with a degree in computer science will go a long way towards establishing society's trust in information technology resources.

The defensive programming paradigm is as earth-shattering now, at the turn of the 21st century, as the object oriented (OO) programming paradigm was in the late 20th century. Just as the OO method was initially met with resistance, so shall widespread adoption of secure programming practices be slowed by generations of lazy programmers and instructors who prefer to maintain the status-quo. Content to insist that defensive programming is too hard for their students to grasp, these instructors will continue to help produce mediocre programmers bent on performing the least amount of work required to get the job done.

We have borne witness to a new generation of students: a generation raised in a world where sloppy programming and inattention to security has impacted their daily lives.

Guided by a drive to improve on the mistakes of their forerunners, these new software developers are starving for instruction that will enable them to succeed where their predecessors have failed. The instructors of today's basic programming classes must cast off the shackles of laziness and complacency and lead these young minds towards a future bright with the hope of safe and trustworthy computing for all mankind.

We have begun the assault on the shores of poor software development by providing a set of instruction tools and ideas that will introduce students to the defensive programming paradigm. These suggestions and concepts can be easily integrated into the four core computer science classes at UMBC, leaving instructors with little excuse for not incorporating them. In order for our assault to be successful, our ideas and suggestions must be implemented as the first significant steps towards showing what a defensive programming based curriculum could look like at UMBC. These course materials, while specific to UMBC, should still be very applicable to courses at other institutions. There is much work yet to be done in this battle, but we are undaunted. We are encouraged by the instructor response to our ideas, and hope that remaining open problems will be solved by members of our team or others in this field. The future of the world rests in the hands of students learning to program. Providing them a strong foundation that allows them to write safer code will pay dividends for years to come.

6.0 Bibliography

[AN96] Anonymous, "A Lab Engineer's Check List for Writing Secure Unix Code", May 1996. ftp://ftp.auscert.org.au/pub/auscert/papers/secure_programming_checklist

[DD02] M. Dark, J. Davis, "Report on Information Assurance Curriculum Development", June 2002. http://www.cerias.purdue.edu/education/post_secondary_education/undergrad_and_grad/curriculum_development/information_assurance/report_info_assurance_cur_dev.pdf

[DW00] D. Wheeler, "Secure Programming for Linux HOWTO"; available: http://www.theorygroup.com/Theory/FAQ/Secure-Programs-HOWTO.html; Feb. 2000

[NG02] N. Gray, "Class notes for CS1 & CS2—Iterators"; http://www.uow.edu.au/~nabg/ABC/iter.html ; 2002

[JMC94] J. McConnell, "National Training Standard for Information Systems Security (INFOSEC) Professionals—NSTISSI No. 4011", June 1994. http://www.nstissc.gov/Assets/pdf/4011.pdf

[MU02] Mullins, "Panel on Integrating Security Concepts into Existing Computer Courses", March 2002, ACM SIGCE Bulletin, Vol. 34, p. 365

[NCSA] NCSA, "NCSA Secure Programming Guidelines", no date.
http://archive.ncsa.uiuc.edu/General/Grid/ACES/security/programming/

[IS01] I. Sommerille, "Software Engineering 6th edition", Addison Wessley, Chapters 16,17,18,& 21. ©2001

[VA00] Richard Vaughn, "Application of Security to the Computer Science Classroom", March 2000, AMD SIGCE Bulletin, v. 32, p. 90

[VBKM00] Viega,Bloch,Kohno,McGraw, " ITS4: A Static Vulnerability Scanner for C and C++", Dec. 2000, Presented at Annual Computer Security Applications Conference, http://www.acsac.org/2000/papers/78.pdf

[YA01] T. Andrew Yang, "Computer Security and Impact on Computer Education", May 2001, ACM Journal of Computing in Small Colleges vol. 16, p. 233

Evaluated UMBC Course Websites:
CMSC 201 Computer Science I for Majors (C programming)
http://www.csee.umbc.edu/courses/undergraduate/201/spring03/
CMSC 202 Computer Science II for Majors (C++ programming)
http://www.cs.umbc.edu/courses/undergraduate/202/spring03/
CMSC 341 Data Structures
http://www.cs.umbc.edu/courses/undergraduate/341/spring03/index.shtml
CMSC 345 Software Design and Development
http://www.csee.umbc.edu/courses/undergraduate/345/spring03/

6.1 Acknowledgements

We would like to say a special word of thanks to the course instructors interviewed throughout this project: Ms. Block, Ms. Bogar, Mr. Edelman, Mr. Frey, Dr. Oates, Dr. Peng, Dr. Pinkston, and Mr. Raouf. Without their generous input, feedback, and comments, this project would not have been possible.

We would also like to thank Dr. Sherman for his instruction, editing, comments, and guidance throughout the semester while we worked on this project.