Project Progress Report

Project title: A Strategy to Include Defensive Programming Tactics in the Undergraduate Computer Science Curriculum at UMBC
Date: April 9, 2003
Investigator Names:

- Brian Roberts, Graduate Student @ UMBC, roberts2@umbc.edu
- Doug Cress, Graduate Student @ UMBC, cress1@umbc.edu
- John Simmons, Graduate Student @ UMBC, js5@umbc.edu

What we have accomplished so far

Our research project requires us to step out of the role of student and don the role of an instructor. Unfortunately the three of us are somewhat lacking in experience as instructors and felt that our first course of action should be to present our basic research plan to the eight instructors responsible for the three core computer science classes 201, 202, & 341. While meeting with the teachers we solicited suggestions on ways to improve our initial ideas and their thoughts on what the best strategy would be to incorporate defensive programming techniques at UMBC. Surprisingly most of the instructors interviewed felt that incorporating security and defensive programming ideas at the lower level classes was a bad idea. The source of these comments appear to arise from the instructors' lack of education themselves in the danger of insecure code and its impact on society as a whole, and the belief that the students in the beginning classes just couldn't 'handle' security subject matter. Counter to the instructor's resistance we believe that the major selling point of our whole project is that the same basic practices which lead to bug-free programming also lead to secure programming. Additionally we feel that concepts that are reinforced throughout a course of study, stand a stronger chance of actually being retained by students.

As part of the interview process we determined that a consistent set of dangers that we as a team were trying to convey to the students should be presented to the instructors. We choose these concepts because we believe they could be easily understood by the students and would illustrate the seriousness of the concepts they were about to learn.

- The harm that programming errors and poorly secured code cause
  - Physical damage that has resulted due to bad code
  - Widespread malicious cyber-events against companies, gov't, & education
  - 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 left un-validated
    - Can result in unexpected behavior in programs and invalid results

    - Segmentation faults due to accessing outside the bounds of an array
    - Printing data from the wrong memory addresses
        - Can potentially access and send sensitive information
    - Use of insecure functions strcpy(), gets(), etc…
    - 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.

Next we evaluated the basic curriculum of each of our targeted classes searching for the best places to weave in defensive programming concepts. We then created course specific suggestions and ideas that we hoped the instructors could then use to fill in these gaps. Next we interviewed the instructors who were able to provide specific feedback regarding the validity of our suggestions. During the interview process, the instructors provided suggestions of their own on how to best go about incorporating defensive programming practices within the context of their particular class. Please see appendix A for sample questions asked of the instructors.

Beginning with CMSC 201 we briefly interviewed Sue Bogar and Dawn Block regarding their thoughts on adding defensive programming practices to the 201 curriculum. Basically 201 is the introductory computer science course for majors, teaching them how to code in C. Most of the students in this course are freshman with very little formal experience programming or computer science skills in general. According to the instructors, many of the concepts of "defensive programming" might elude their 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 aren't 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" 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 Sue and Dawn were a bit reticent when it came to altering the curriculum primarily because they felt that the students were faced with the very overwhelming task just of

learning the material already included in the curriculum. even 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 security belonged in some course, but neither wanted it in theirs.

One of the first steps that the instructors thought would be beneficial would be 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 purpose, there is already great stress placed on I/O validation. Further emphasis could be illustrated 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() which they typically mention in lecture already and then ask the TA's to go into greater depth during the discussion section.

One of the greatest obstacles that we face in 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. The problem with this approach is that the notes change every semester, and there is no standardized format for notes. The discussion section notes are merely required to discuss lecture topics in greater depth, resulting in TA's with particular strengths and interests incorporating those 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 might 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 we convince the instructors to incorporate "defensive programming" practices into their lectures, we would have to constantly reiterate the point with the new TA's every semester.

Next we interviewed the two instructors teaching CMSC 202 – Mr. Raouf and Mr. Frey. 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. Absorption 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 is apparent that small amounts of new material are easily added to the lectures. Pointing out the relevant security pitfalls as new

topics are covered will be 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. It would 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). Mr. Raouf suggested that a project regarding check digits for credit card numbers might suggest a good security theme. Including defensive programming concepts in the grading criteria, however, would not be possible. For example, the projects already list user input validation as a gradable criteria, 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.

We also gained some insight into the content we should include for the class. Students in this class should be able to understand concepts referencing 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 OS 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.

Finally we interviewed the four instructors that take turns teaching CMSC 341 – Data Structures: Mr. Frey, Mr. Edleman, Dr. Peng, and Dr. Oates. Currently Dr. Oates is serving as the course coordinator. 341 has approximately 29 class meetings where the 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. ADTs that describe how to manipulate stacks, queues, trees, heaps, hashes, and graphs are covered in class and reinforced through 5 programming projects assigned throughout the semester. Additionally the students' understanding of the concepts taught in the class are evaluated by 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 talking at them during lecture. 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 security related test question on one or more of the exams. Lastly we asked the instructors where they thought the best place was to teach defensive programming and computer security related topics within the UMBC computer science curriculum.

Dr. Oates felt initially that secure programming concepts might be adding more work to the students' plate then they can handle. He pointed out that most of the students consider 341 to be a challenging class and that by adding additional security requirements to their programs/projects might be overwhelming. After further discussions 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 techniques. Each of the projects assigned in 341 has 2 -3 questions that must be answered by the students relating to the project. These questions are designed to make the students think about what they have done and why they solved the problem the way they did. Dr. Oates thought that a question about something relating to security/defensive programming would fit nicely here. Additionally Dr. Oates felt that the best places to include defensive programming tactics was early on 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 as well. Finally, Dr. Oates was adamant about not including security related questions on any exam.

Mr. Frey 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, Mr. Frey said 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.

Mr. Edleman was a wealth of suggestions and input. His initial concerns were very similar to Dr. Oates' that the students taking 341 are pretty 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 don't really seem to be very important to them and therefore aren't given the serious consideration they deserve. Mr. Edleman 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 Dr. Oates, Mr. Edleman 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 Mr. Edleman'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. Mr. Edleman 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 Mr. Edleman 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 Dr. Pinkston the department head to see what his opinions were on the matter.

The last professor we interviewed for 341 was Dr. Peng. Like all of the previously interviewed 341 instructors, Dr. Peng felt that 341 was perhaps not the best possible fit for defensive programming concepts. However, Dr. Peng 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. Dr. Peng 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. Dr. Peng stressed that any slides offered to the instructors should be concise and easy for them to understand. Finally, Dr. Peng felt that students taking 202 would not be interested in security concepts at all.

Following the suggestion of several of the instructors we also interviewed Dr. Pinkston the UMBC CS 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 vice 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 direction to head in. 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 the subsequent relevance of which to buffer-overflows. He felt that CMSC 345 (Software Design and Development) was an excellent capstone class for defensive programming. The students in this class are split up into groups and then assigned a customer for whom they must develop a software package. This package requires the interaction of several components including internet related applications and local end users. 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. He also mentioned that if a majority of the department felt that security issues should be included in the curriculum, then the individual resistant instructors would probably capitulate.

We have sought several testing methods in an effort to add more rigor to the evaluation of our research. Unfortunately the time constraints placed on this project will limit our aility to apply any of them. Our first testing solution would be to conduct surveys by sampling students at various levels within the CS curriculum to ascertain their current experience and knowledge of security and defensive programming issues. If we had the time we would like to ask a sampling of students at various stages in the UMBC curriculum the same basic questions. We would then, starting with the fall semester, begin to include our suggestions and changes to the curriculum. Then at the beginning and end of every semester we would survey a random sampling of the students 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 but would likely 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 and buy in by the department and the various class instructors. Though this method might clearly show the level of comprehension the students had attained, the amount of work required by the instructors could be considered prohibitive from their point of view.

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 take a real-world project specification, develop a software solution for it, and then submit it for a security audit before the contest judges. 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 proceeded 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 shortfalls of such an evaluation technique would be the difficulty in proving the impact of our educational ideas on the students work, vice 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. Unfortunately as with the other evaluation techniques, we the investigators would not have the time to implement such an idea within the bounds of time left for this project.

What remains to be accomplished

We have accomplished a great deal of work towards completing our proposed research goals. However there are still details of the project that we have not yet addressed completely. Since we have decided to incorporate CMSC 345 as our capstone defensive programming class, we need to interview the instructors for that class and evaluate its current curriculum to determine where the best fit for our supplements would be. Additionally we have decided to tone down the amount of influence we wish to exert on the lower-level classes (201,202) and therefore we need to spend some more time fleshing out exactly what material is worth being covered in those classes versus being saved for later inclusion in other classes. We still need to develop a project specification for 341 that will incorporate more defensive programming aspects. And lastly we need to determine whether or not we can evaluate our curriculum suggestions within the time left this semester.

Difficulties encountered and solutions to them

We were surprised by the intense resistance from most of the instructors for the various classes. Most of them felt that security and defensive programming were good things but such concepts didn't belong in their classes. As we have gathered input from them, we have looked for ways to present our idea and convince the instructors of the necessity to include them in their curriculums. We have decided on three solutions to solve this resistance problem. First we will make our curriculum additions as simple and concise as possible to allow the instructors to incorporate them as easily as possible into their syllabi. Secondly we will provide some minimal education for the instructors themselves consisting of a couple of slides that describe the dangers of insecure programming and what some examples might look like, including a buffer-overflow example. Lastly we will work with department leaders in order to lobby them on the importance of including defensive programming into the UMBC CS curriculum.

Our second major difficulty is the long term nature of the project itself. When one is trying to influence a group of people across the length of their educational experience, one needs to interact with them throughout that whole time. Unfortunately our project

timeline only encompasses a single semester. With such a short timeframe within which to study our problem, provide solutions, and then study the results we feel that we will not be able to adequately quantify the effectiveness of our ideas. We have proposed several suggestions of how we would like to go about testing our results but at this time, we don't see any particular solution to the problem.

Significant changes to proposal

Our project has undergone some significant changes since we first proposed it. The first of which was the name change of the project. After meeting with several of the professors we believed that a focus on secure programming was not as effective as a focus on defensive programming. This has a little softer sound and will hopefully decrease the amount of resistance that the instructors have thus far put forth. Additionally we feel that such a focus allows the students to be more marketable when they seek employment outside of the academic environment. Finally a curriculum-wide focus on defensive programming allows UMBC to be more marketable to adult students seeking to specialize in security.

Our second major change to the project was again predicated by the advice of the professors that we have interviewed thus far. Many of them felt that focusing on security at the lower level classes (201, 202, & 341) was too much for the students that were taking the classes. In response to this we have decided to take more of a graduated approach towards teaching the students about security. By introducing a few concepts in 201, then a few more in 202, and yet more in 341, and with a final capstone lecture in 345, we allow the students' comprehension of security issues to grow as their understanding of computer science in general grows. We felt that it was very important to target the programming classes that all computer science majors were required to take in order to have the broadest impact. Focusing on just a 400 level security class offered as an alternative would not be sufficient for the kind of impact we are hoping to drive home to the students.

Finally, due to feedback on our initial project proposal about a lack of a significant research component, we have decided to interview the instructors both before we developed materials and afterward. This new procedure will help to validate our instruction methods and lend more legitimacy to our results.

Draft outline for final report

I. Introduction
    A. Abstract
    B. Why does the undergraduate community need more security training?
    C. Summary of results
II. Description of our solution
    A. Our initial approach to incorporating security into curriculum

B. Initial negative instructor response
C. Our resulting focus on defensive programming and multistage agenda
III. Proposed testing evaluation methods of our solution
A. Survey procedures
B. Alternative ideas
IV. Conclusions
A. Remaining open problems
V. Bibliography
VI. Appendix
A. Sample questions posed to instructors
B. Resulting materials provided to department
1. PowerPoint slides
2. Project specifications for 341 & 345

Updated 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

[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/

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

[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 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/

Revised Schedule

March 19	Proposal
March 21	Evaluation of a peer proposal by email
March 28	Develop list of topics
March 31	Begin Interview of instructors
April 4	Develop coverage goals
April 8	Finalize midterm
April 9	Midterm delivery
April 10	Finish first interview round of instructors
April 11	Develop all course materials
April 14	Begin second round of Faculty interviews
April 18	Finish second round of Faculty interviews
April 21	Pre-draft of paper and presentation
April 23	Complete draft report and draft presentation for review
April 30	Referee report of a peer project
April 30	Oral presentations begin
May 7	Final report

Appendix A – Interview Questions

Sample Questions/suggestions for places to incorporate security into CMSC-341
I notice that there's lecture space in the syllabus for an 'advanced topics' lecture. Would a single lecture on security at the end of the semester be worth while? Or would it be better to integrate security related slides throughout the course?

There appears to be 5 projects spread throughout the semester.

- Would a "security hazards that this data structure is prone to" section be appropriate for each project on its description page?
- Would a security oriented question be appropriate as a third question (or as a replacement) on the question page for each project?
- Would a security requirement for each project description be a gradable element?
    - i.e. All input should be checked
    - all array bounds must be protected
    - use of the more secure functions and libraries must be included
    - possibly use a tool to statically check for dangerous function calls (ITS4, LCLint, Purify etc..)

- Would it be instructive to pick a student's submission at random and remove any identifying characteristics and then show it to the class pointing out the security flaws in it? Of course the author would recognize his/her work, and therefore everyone in the class would be on their toes so that their program is not shown.
- Would a change to one of the project descriptions to one with more of a security flavor be useful? Of course the project's focus would continue to be the data structure intended to be taught according to the syllabus.

There appears to be 3 exams. Would it be possible to add a security related question to each or any of the exams?

If you were to incorporate security into the core of UMBC computer science education would you include the topic in 341? If not where would you place such instruction?