Proceedings of the 13th Colloquium for Information Systems Security Education
University of Alaska, Fairbanks
Seattle, WA June 1 - 3, 2009

Computer Security-focused Programming Assignments in Foundational CS Courses

Kara Nance and Brian Hay, *University of Alaska Fairbanks*

Abstract – While the emphasis on computer security education within specialized courses is easy to justify and achieve, it is much more challenging to introduce these concepts across the computer science curriculum to begin to "change the culture" of computer science students in order to create a foundational appreciation for and understanding of computer security issues. This paper describes some techniques that have been applied in early computer science programming courses at the University of Alaska Fairbanks to facilitate computer security education among beginning programming students through the use of computer security-focused programming assignments. This provides a mechanism for strengthening computer security skills within the scope of the traditional course content to foster an awareness of information assurance concepts.

Index terms – Information assurance curriculum, CS1, security education

I. Introduction

The need for incorporating computer security education into the computer security curriculum is widely recognized. There is debate over whether a depth-first or breadth first approach will work best and also over how to add more to the curriculum without losing content in other areas. The Computer Science Curriculum Review Taskforce identified the emergence of security as a major area of concern" and reported that it "has received urgent requests from industrialists requesting that substantial attention to security matters be regarded as compulsory for all computing graduates.[1]" This paper describes a series of security-related assignments that were developed in response to this need. The assignments were used in a traditional Computer Science I (CS1) course in order to increase security-awareness in freshmen CS students within the current curriculum.

Kara Nance and Brian Hay
Department of Computer Science
University of Alaska Fairbanks
210 Chapman
Fairbanks, AK 99775-6670
ffkln@uaf.edu and brian.hay@uaf.edu.

II. Course Structure

The following describes the course structure and details for the Computer Science I (CS1) course from which these examples have been taken. The materials should be easily adaptable to a wide range of course structures.

The course approach is control structures through objects (using C++) rather than early objects [2], although the same concepts have been used in an early-objects course using Java. The average class size is around 20 split between Computer Science and Computer Engineering majors. The set-up is for a traditional lecture course, with an option to meet in a laboratory environment for hands-on experiences when instructor chooses. All programs must be run for the instructor or lab assistant and there are specific hours in which the students in this class have priority in the lab and are guaranteed that their course lab assistant will be present for run checks and questions. The course also uses specialized materials and software available in the Advanced System Security Education, Research, and Training (ASSERT) Lab, although those materials are only referenced in the following assignments and not necessary for any of the following.

The course is 15 weeks, meeting for three one-hour sessions. The CS1 students have priority in the laboratory twice weekly for five hours periods including the five hours immediately preceding their weekly assignment due date and time.

In order to better prepare students for future classes, many of which have semester-long group projects, the assignments have a variety of formats to provide interaction between students. Some exercises are assigned on an individual basis, with each student required to complete all components. Other exercises are partial group assignments, where students complete unique parts of programs and then are assigned into groups to put the components together into one cohesive program. There are also group projects where students work together through all parts of the software development life cycle. Some group projects are self-selecting. Other groups are assigned by the instructor. In addition, there are some competitive assignments that provide students with an opportunity to exceed

ISBN 1-933510-96-9

requirements and then are subsequently awarded additional points based on the level of success.

Our institution is committed to a single instructor taking students through the CS1/CS2 sequence. The distribution of assignments between individual and various formats of group assignments provide the instructor with a chance to get to know the students and identify their strengths and weaknesses and then to work with them to improve their overall performance.

While the previous is a description of the course structure for our institution, it is not required in order to use the following (or similar) assignments. It is provided so that examples may be taken in context, but materials should be easily adaptable to a wide range of course structures, languages, and models.

III. ASSIGNMENTS

The first few assignments are relatively simple assignments to ensure that the students are on track, but even at this early stage, programs can have a subtle computer security emphasis. Assignments generally take the form of a memorandum to the students from an individual that includes a background story and a resulting task. An example memorandum regarding backups is shown in Figure 1. This requires students to identify the characteristics that are important to solving the problem. Frequently the computer security education component is hidden in the associated "back story". The outline for each of the following examples will include a list of the programming concepts that are new to the students for use in this assignment. Note that the lists are not exhaustive, but meant to indicate the limitations within which the students are working. The course syllabus restricts students from using libraries and commands not yet covered in class without permission (for example, they are required to use C++ arrays rather than the STL vector class to ensure that they understand the concept of arrays.)

Memorandum

To: Programming Team

From: I.M. Theboss

Date: 9/27/2008

Re: Recent Server Crash Without Backup

As you know, I.R. Untrained, the new system administrator, had not been backing up the system files as outlined in our Company Security Plan. When the hard drive failed on Thursday, many of our programming efforts were lost. Fortunately, the VERY VERY expensive computer forensics company we hired was able to recover some of the content from the drive using Access FTK, a forensics package that they borrowed from the ASSERT Lab at the University of Alaska Fairbanks.

The attached file contains the recovered material. I want to remind you that we have a deliverable that is due to our client on Monday, September 22 at 1pm. Please work with your team to ensure that this deadline is met.

Of course, the code is just in a single text file now and will need to be separated out into separate files for modification. The code may not be complete. I also noticed, when previewing this code that it appears the programmers were unaware of the C++ iomanip library. I encourage you to take advantage of this library to make the code and the corresponding output more readable.

I suggest that each team member take one of tasks 1, 3, and 4 and that you all work together to approach task 2.

Good luck with this task and remember to always back up your data.

Figure 1: Example Assignment Memorandum

The following set of examples represents only a small portion of the computer security-related assignments that the students are required to complete during the semester. They were selected as a representative sample to demonstrate a range of computer security concepts that can be incorporated into CS1.

A. Example 1 – Logfile Evaluation

This assignment introduces students to the concept of a log file and the several types of log files that can be produced. Example log file entries are provided with (in some cases) simplified variants for fields within the log file. Students are required to write a program that calculates the space requirements for various types of log files.

1. Language Components:

At this point, students have been introduced to the native data types, the concepts of variables and literals, the arithmetic operators, comments, and very simple program structure. This assignment will provide them with the opportunity to demonstrate mastery of these concepts. They have all written and run a simple "hello, world" program for the instructor or lab instructor and thus have demonstrated basic competency with their programming environment. Students do not yet have input capabilities, so all information is included in the problem description, with the exception of the values that they must calculate or determine.

2. Problem Description

Students receive this assignment as a memorandum from the system administrator (their boss) who is planning the storage requirements for the log files that the company has decided to track as part of their Computer Security Plan. Faced with the need to order appropriate disk storage space and backup space, the system administrator needs to determine the memory requirements associated with each type of record. An example abbreviated problem statement is provided below. The associated log entries are shown in Figure 2.

NetFlow Example Log (Simplified)
Destination Interface: 7 characters
Destination Prefix: Four integers separated by "."
Network Mask: "/" followed by integer
Flows: Integer
Packets: Integer
Bytes per Packet: Integer

Dst	Dst Prefix Msk	Flows	Pkts	B/Pk
Et0/0.1	172.16.6.0 /24	5	13	52
Et1/0.1	172.16.7.0 /24	3	31000	1314
Et1/0.1	172.16.1.0 /24	16	104000	1398

Figure 2: Example Logfile

Given the following log formats, the students are expected to calculate how much memory will be required to store one record for each second of a day for each format. In order to accomplish the calculations, they need to know how many bytes the various data types use, which they can determine from within the program, their text, or from system documentation. The students are expected to write a C++ program that will determine and report the amount of memory required to store one day's worth of log entries for a given format.

3. CS Objectives

The intent of this exercise in the scope of the CS1 class is to provide students with an exercise that demonstrates their understanding of basic arithmetic operators in C++.

4. Security Objectives

This exercise is intended to introduce students to the concept of log files, including the different types and formats of log files generated by the operating system, applications, and network devices. This also offers some insight into the concept of a digital footprint, in which evidence of activities on a computer system can be gathered from a variety of sources.

5. Observed Outcomes

Students were generally unaware of the amount of logging that can be generated associated with computer usage. Discussion included network-related logging as well as computer-based logging such as file modifications, accesses, etc. This assignment generates lots of questions about log files, log file formats, who is logging what, who decides what is logged, etc., which lead into policy decisions as well as digital forensics discussions. The results are particularly interesting when different students or student groups are assigned different log formats and present their formats in class.

6. Variants

There are several variations to this exercise that can be used in CS1 or CS2 classes, including the following:

- Different log file types can be assigned to each student or group of students.
- The students can be given a maximum disk space, and be tasked with determining the maximum number of log entries that could be stored in that space. If they are also given a rate at which log entries are added, they can determine the amount of time before logs will be deleted (or sent to a backup device)
- Students can be asked to determine if the types used in the example log files are appropriate, and if not whether a more efficient approach could be used. For example, a date/time could be stored as ASCII text in yyyy-mm-dd:hh.mm.ss, using

19 bytes, whereas the same information with a limited date range could be stored using a 4-byte int. They could then recalculate the storage requirements based on this new format.

In addition to the arithmetic operator exercises, log files are also useful for testing comprehension of other topics, including the following:

- File IO assignments in which students read and write log files, both in ASCII and binary formats.
- Search exercises, where students need to search a log file for entries that are relevant to some search criteria (e.g., all entries that include a reference to a given source IP address).
- Sorting exercises, where students are asked to either sort, or generate a sorted index for, a given log file. For example, a log file may be sorted by time as it is read from disk, but students may be required to generate an index for the source IP address in the log. This exercise can clearly be combined with an associated search exercise.

B. Example 2 – Password Policy

This assignment uses principles of combinatorics to test the mathematical robustness of a password.

1. Language Components:

At this point, the students have not been introduced to loops, but do have I/O capabilities and programs can interact with a user.

2. Problem Description

Students are required to calculate the number of unique passwords that can be generated using different password rules. The program must ask the user for the length of the password and then asks for the characteristics associated with each value in the length. Choices for categories include lowercase letters, uppercase letters, lower and uppercase letters, digits, special symbols, and all of the above. (The list of acceptable special symbols is enumerated.) The program then outputs the number of unique passwords that can be generated using that rule. There is a set of 5 test conditions that students must run, then they find five other password rules that meet a specified criteria. For example:

If a password cracker can test 100,000 passwords per second, what is the minimum length that a password consisting of alternating digits and lower case letters must be to ensure that all passwords cannot be checked prior to changing the password every 30 days?

3. CS Objectives

The intent of this exercise in the scope of the CS1 class is to provide students with an exercise that demonstrates their understanding of interactive I/O, variables, and arithmetic operators.

4. Security Objectives

This exercise is intended to introduce students to several concepts associated with passwords, including what it means for passwords to be considered strong or weak. It also demonstrates the extent to which additional characters or added password length reduces the susceptibility to brute force attacks, and how passwords can be chosen in a manner that ensures that they are both hard to guess and (relatively) easy to remember. The exercise also generally prompts some discussion of common (and easy to guess) password, why different password should be used for each account (or at least each group of accounts), and some methods by which multiple passwords can be remembered or stored safely.

5. Observed Outcomes

This assignment is a real eye-opener for students, especially when coupled with a password cracking demonstration. Students were generally surprised by the effect of increasing password length on the robustness of the password. It also generated some discussion of social engineering and using associated methods to determine a password.

6. Variants

There are several variations to this exercise that can be used in CS1 or CS2 classes, including the following:

- Run the password lists on a password cracker to demonstrate dictionary attacks.
- Extend to discussion of attack strategies.
- Extend to social engineering and discussion of types of passwords people tend to use and why they are strong/weak.

C. Example 3 – Recovering Deleted Images

This assignment provides students with the opportunity to recover images that have been deleted from a device.

1. Language Components:

Students now have looping and file I/O capabilities and can extend earlier concepts and apply them to files.

2. Problem Description

Students use a recovered disk image and search the file for the JPEG header and footer markers and write each detected JPEG file to a unique output file. The disk image is typically created using the *dd* utility on a digital camera memory card on which several JPEG images have been placed (i.e., a clean card was inserted in the camera, and several pictures were taken). The use of the digital camera memory card is effective as images are often placed sequentially on the disk, as opposed to being

subject to fragmentation on other media, such as a hard disk drive taken from a PC. Prior to creating the disk image with *dd*, some of the JPEG files are deleted from the memory card (either using the digital camera interface or by connecting the memory card to a PC).

3. CS Objectives

The intent of this exercise in the scope of the CS1 class is to provide students with an exercise that demonstrates their understanding of file input and output, and the ability to search for specific patterns within a data stream using character and string manipulation.

4. Security Objectives

This exercise is intended to introduce students to the field of digital forensics, which is a subject that UAF covers in greater depth in specific 400 and 600 level courses.

5. Observed Outcomes

This exercise raises awareness of how files are deleted from storage devices, and that file data may remain easily accessible even after it the user believes it has been deleted. The discussion extends to image formats, ASCII representation, and string libraries.

6. Variants

There are several variations to this exercise that can be used in CS1 or CS2 classes, including the following:

- A (small) hard disk image can be used in place of the digital camera memory card as this raises the issue of fragmentation, and as such students may encounter a second JPEG start of image marker without having found the end of image marker for the first image.
- Students can be tasked with finding instances of keywords within a disk image, and saving each block of data that contains keywords.
- These exercises can be combined with a "scavenger hunt" in which clues to locations to be visited are placed within deleted files on a small disk image. (While this may involve more time than is typically available in a traditional CS1 or CS2 class, it has been particularly well received during summer camps for high school students.)
- Students can be tasked with locating malware signatures (e.g., a specific byte sequence) in network packet traces. The network data in this case I generally provided in a file, rather than requiring the students to monitor an actual network connection.
- Students can be tasked with writing an implementation of the *strings* utility, which reports all printable ASCII sequences of a given length or longer in an input file. This is particularly fun when an executable file containing a username and password is distributed as part of the assignment, with the goal of having the students use their *strings* implementation to find the user name and password which "unlocks" the executable.

D. Example 4 – Basic Cryptography

This assignment provides students with the opportunity to encrypt and decrypt data using several basic encryption algorithms.

1. Language Components:

Students now have looping capabilities and an understanding of strings and string manipulation.

2. Problem Description

This exercise typically consists of two components:

- Encryption: students are tasked with writing a program which accepts some input string (i.e., the plaintext message) and, depending on the algorithm being used, an encryption key. The program should produce the appropriate ciphertext as output.
- Decryption: students are tasked with writing a program which accepts some input string (i.e., the ciphertext message) and, depending on the algorithm being used, a decryption key. The program should produce the original plaintext as output.

Once they have written both the encryption and decryption components, students encrypt a message under a key, then pass the key and the ciphertext to another student who performs the decryption (and confirms that the plaintext they generate is correct).

This assignment typically proceeds in several stages which escalate in difficulty with the goal of challenging the more advanced students while ensuring that the exercise is accessible to the entire class. An example of such a sequence is:

- Morse Code encoding of text (no key required)
- Encryption/Decryption using a simple substitution cipher in which each character is replaced by the character *n* places to the right in the alphabet (e.g. the Caesar Cipher). In this case, the encryption key is the single value *n*.
- Encryption/Decryption using a more general simple substitution cipher. In this case the key is the substitution sequence to be used (e.g, A→R, B→E, etc)
- Encryption/Decryption using a polyalphabetic substitution cipher, in which the key is a series of simple substitution keys.

3. CS Objectives

The intent of this exercise in the scope of the CS1 class is to provide students with an exercise that demonstrates their understanding of loops and string operations.

4. Security Objectives

This exercise is intended to introduce students to the field of cryptography, which is an important topic for many aspects of information assurance. While the encryption algorithm used in this exercise is insufficient for real use (as is made clear to the students), it does provide an initial introduction to the field.

5. Observed Outcomes

This exercise leads to interesting discussion about cryptanalysis and associated techniques. It has been used as an introduction to algorithm analysis as various student solutions can be discussed and analyzed.

6. Variants

The most common variant of this exercise involves the addition of a third component, in which students are given a ciphertext created using a simple substitution cipher but no key. They are then tasked with writing a program to help them decrypt the message by calculating and reporting the letter frequencies in the message, which the students can then use to solve the problem of recovering the plaintext message. A more advanced version of this exercise involves the program also calculating and reporting the digraph and trigraph frequencies in the message.

E. Printer Forensics

This assignment has been used as the final programming assignment for the CS1 course and the students consistently rate it as the most interesting assignment. The assignment involves the investigation of laser printer watermarks.

1. Language Components:

At this point in the semester students have been exposed to functions, structs, classes, 2-D arrays, loops, and file I/O.

2. Problem Description

An in-class demonstration using black lights and a variety of printouts on high glass paper is used to introduce students to the concept of laser printer watermarks. Students are then given an image similar to that shown in Figure 2.

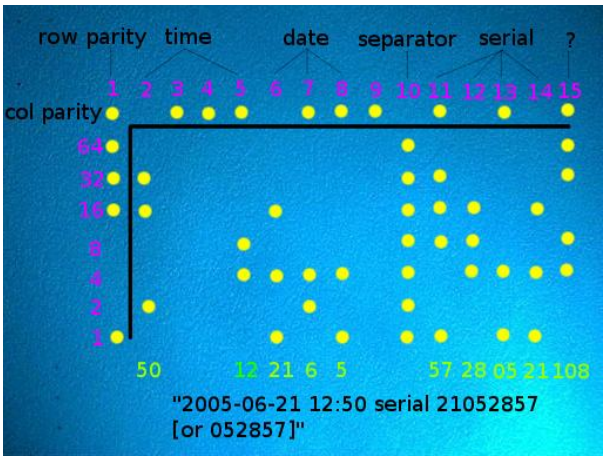

Figure 3: Example Printer Watermark [3]

This figure is accompanied with a detailed breakdown of the meaning behind the dots for this particular printer model. Students are then required to write a program which includes functions to "decode" two dimensional dot formations into structured information and "encode" structured information into the associated two-dimensional dot array.

3. CS Objectives

The intent of this exercise in the scope of the CS1 class is to provide students with an exercise that demonstrates several concepts they encountered during the semester, including file I/O, arrays, and functions.

4. Security Objectives

This provides an introduction to digital forensics, data encoding and decoding, and approaches to combat counterfeiting.

5. Observed Outcomes

This is an extremely popular assignment that generates much discussion both in and out of class. Students are amazed to see that an apparently white page that has been through a laser printer is actually covered with yellow tracking dots. Discussions extend to security policy and the role of the U.S. Secret Service in this effort.

6. Variants

There are several variations to this exercise that can be used in CS1 or CS2 classes, including the following:

- Students could be given the task of developing their own dot configurations (i.e., encoding schemes) for encoding a given set of data, such as time, data, printer serial number, and printer IP address.
- Students could be provided with a series of dot sequences taken over a time range with the goal of having them attempt to determine what the dot patterns meant (i.e., what encoding scheme was

used). Students could be given some relevant information, such as the printer serial number, IP address, date and time range, and printer firmware version number (some of which may not be part of the encoding). The goal would be to write a program based on the information provided which decoded the dot sequence for the identified encoding sequence.

IV. SUMMARY

Using security-focused exercises in beginning programming classes has allowed instructors to increase student awareness about security issues at the beginning of their educational experience rather than introducing it during specialized courses at higher levels. In addition, there are non CS programs that require the CS1 or CS1/CS2 sequence. This allows us to introduce the concepts to students in other majors also. The assignments allow students to demonstrate mastery of the traditional CS1 skills being presented in class and to apply them to real-world scenarios. Students enjoy the "real-world" scenarios and also become aware of the opportunities for specializing in computer security as they pursue the major. To meet the needs of the students and to foster even further discovery, the ASSERT Lab is developing self-study modules to provide further information about some of the follow-on discussions that result from these assignments. Overall, we have found that security-focused programming assignments in early CS courses allows the early introduction of computer security concepts within the scope of a traditional CS1/CS2 sequence while exciting students and making them security-conscious.

V. REFERENCES

[1] Computer Science Curriculum Review Taskforce. Review of the CS2001 undergraduate volume. http://wiki.acm.org/cs2001/index.php

[2] Gaddis, Tony. Starting out with C++: From Control Structures through Objects, sixth edition. Pearson Education, Inc. 2009.

[3] EFF Website. Retrieved 12/01/08 from http://w2.eff.org/Privacy/printers/docucolor/