Student Involvement in Designing a Network Encryption Algorithm

Ronald Marsh, Ph.D.

Department of Computer Science & Operations Research
North Dakota State University
Fargo, ND 58105
rmarsh@plains.nodak.edu

Abstract

In an effort to expose computer science students to the issues surrounding computer security and computer science research topics, freshman level computer science students were assigned the task of designing and implementing an algorithm for encrypting ethernet packets. The goal was to develop an encryption method that could be used to automatically encrypt all ethernet packets at the network or transport layer.

Several assumptions were made to insure that the task would be appropriate for a freshman level class. The most significant assumption was that the encryption method would only be required to provide "minimal security". The second most significant assumption was that students could simulate the network by reading and writing text files.

In addition to a review of the designs submitted by the students, we also present results of a survey taken by the participating students. The survey provides feedback as to what the students thought of the task, if they felt they arrived at a successful solution, if they would like to do something similar again and if the task had changed their perceptions of computer science.

Introduction

To provide early exposure to the issues surrounding computer security, to software engineering and to computer science research topics, freshman level computer science students were assigned the task of designing and implementing an algorithm for encrypting ethernet packets. As a take-home final exam, the students were required to document the design, the test cases applied and to take a 7 question anonymous survey (the survey was done separately from the take-home exam).

The task assigned fit well into the overall "flavor" of the class that semester. Several assignments involved the writing of simple encryption programs. Students were also kept informed about the latest computer security hazards and attacks directed towards campus computer systems. Students were also taught that security is not absolute and that there is not one standard that will fit all businesses and industries[1]. Therefore, as long as some general requirements were met, students were allowed to pursue any encryption approach they desired (block ciphers, stream ciphers and public-key methods were covered in class).

Since none of the students had yet taken a software engineering course, the assignment indicated that the majority of the effort should be directed towards algorithm and code development. The algorithm and code development lent itself well for student involvement. Reasons include:

1. The application assumptions kept the overall project complexity low. Therefore, the task could be completed in less than one semester.

2. Enough implementation variations existed to challenge each student according to their own abilities.

3. Implementing the algorithm(s) developed required most of the programming constructs covered in the class (the student's first course in C++).

The software engineering portion of the project (the take-home final) was intended to re-enforce three concepts.

- The importance of documenting one's code in a manor such that another does not have to actually read through the code to verify that it meets the design goals.

- The importance of designing/developing good test cases and using those test cases to validate the code.

- That computer science is more than "just cutting code".

The survey was used to provide feedback as to whether or not the students found the project interesting, was something like would like to do again and if the project had changed their perceptions of computer science.

Background

As corporate and private America does more and more of its business over networks the issue of network security shifts from an ethical debate to a financial imperative. Considering the growing need and demand for providing vender type services over the network and the high degree of skill and tenacity exhibited by many hackers, the need for more robust security measures continues to increase. The cable television market is a prime example of this as an entire underground industry has arisen which provides people with devices to obtain cable television services without paying for them. One can only expect that as more vender type services are offered over the internet, these underground industries will target the internet as well.

The current state of network/computer security technology is to use one or more of the following techniques.

- Use complex encryption techniques that are very difficult to defeat directly[2].

- Embed the security protocol into networks at low levels to reduce the number of vulnerable points that the unencrypted data is exposed to with the outside world[2].

- Use a knowledge base of users (user locations, privileges, and owned files)[3].

The first two items can coexist using a variety of techniques.

- Complex encryption techniques supported in software.

- Complex encryption techniques supported by an accelerator card[4].

- Complex encryption techniques supported by an external hardware device[5].

However, hackers have demonstrated again and again that intrusions are always possible. Powerful search techniques have been used using multiple distributed computers connected over the internet to break encryption keys[6]. Special purpose hardware has been built which can break a DES 56-bit key in as little as a few days[7]. Other methods (timing attack, chosen plaintext attacks, common modulus attack, low encryption exponent attack and low decryption exponent attack) have also been used to break public-key encrypted messages[8]. Many less sophisticated methods, such as running a "network sniffer" on a networked computer, are used to collect logins and passwords.

At the university level, complex encryption techniques may not be required or may not be cost effective for use with the general student population. An efficient and low-cost encryption method embedded into the network at a low level may be sufficient to prevent intruders from obtaining logins and passwords with "network sniffer" programs. Such a method would also provide email, ftp and other users with some security as well. Enough security, perhaps, to meet the needs of most students. In an effort to achieve this goal, the students were given the task of designing a method to encrypt the data segment of an ethernet/802.3 frame (Figure 1) and to implement the algorithm in C++ code.

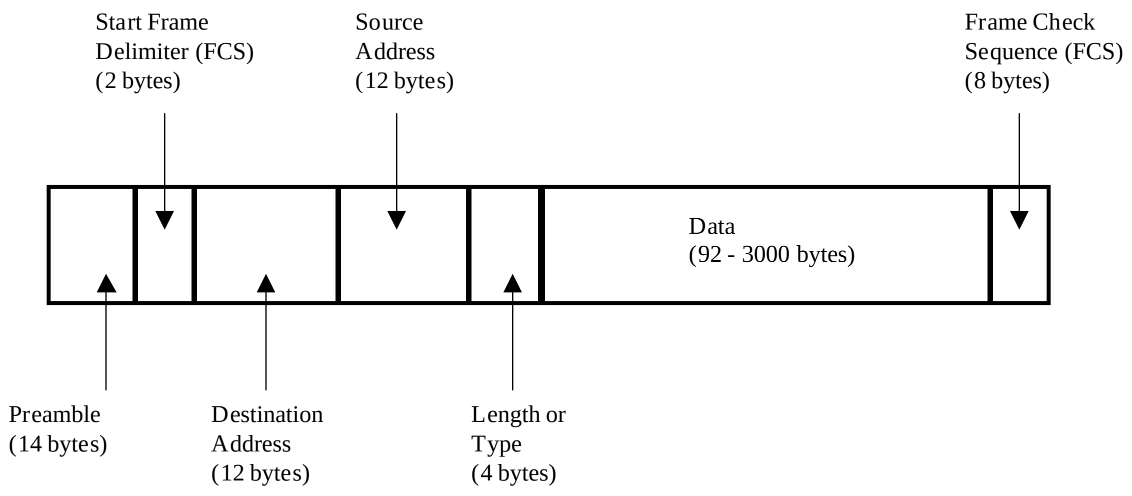
Start Frame Delimiter (FCS) (2 bytes)
Source Address (12 bytes)
Frame Check Sequence (FCS) (8 bytes)
Data (92 - 3000 bytes)
Preamble (14 bytes)
Destination Address (12 bytes)
Length or Type (4 bytes)

Figure 1. Ethernet / 802.3 frame structure.

Additional requirements were also placed on the design / implementation. These are:

1. Efficient - Since the encryption method is to be implemented at a low level, every ethernet packet produced gets encrypted and every ethernet packet received gets decrypted.

2. Eliminate or mask runs of the same characters - Runs of one character type (i.e. blanks) can provide the intruder with clues as to what the packet may contain.

3. The key (if required) was to be generated using the destination address, the source address, and the frame length or type information. Using these three frame segments, a key

schedule containing 28 unique 8-byte keys for each source-destination and frame type/length is easily produced.

4. Ethernet frames were simulated and supplied to the students as text files. The destination and source addresses, the frame length and the date segment were included in each text file. The data segment length varied from file to file. The student's programs were required to read (from disk) simulated ethernet frame files of any length, encrypt the data segment and write (to disk) the resulting simulated encrypted ethernet frame files. The files written by the student's programs were required to have the same format as the simulated ethernet frame file.

5. The student's programs were also required to decrypt simulated encrypted ethernet frame files encrypted with their own algorithm.

Student Designs

During the semester, and prior to the assignment of this task (or research project), several encryption methods were introduced in class. These included block ciphers, stream ciphers, and public-key ciphers. Block ciphers transform a fixed-length block of data into a block of encrypted text of the same length. Stream ciphers operate on streams of data, usually individual bits or bytes. Public-key ciphers can be implemented as either block or stream ciphers. Student algorithms meeting the full requirements were either stream ciphers or block ciphers. Most of the student algorithms were stream ciphers. Only a couple of students developed block cipher algorithms. Since the students were aware of the computational cost of public-key ciphers, none of the students used that approach.

All of the stream cipher algorithms developed generated the key schedule from the 28 bytes forming the source and destination addresses and the frame length or type. In every algorithm encryption was accomplished by XORing the key schedule with the data segment. Figure 2 shows the basic algorithm. A list providing a brief synopsis of acceptable algorithms follows Figure 2.

```
i = 0;
j = 0;
while (i < text_length) {
    text[i] = (key[j] XOR text[i]) + M;
    i++;
    j++;
    if (j >= N) j = 0;
}
```

Figure 2. Simple XOR stream cipher algorithm.

List of Algorithms

1. Use the 28 bytes as 28 separate keys (N = 28). A variation on this idea included the addition of a constant (M) to each key-data XOR result.

2. Combine pairs of the 28 bytes forming 14 keys (N = 14). This is done by either adding their ascii (integer) values or by XORing byte pairs.

3. Using a set sequence, select 8 of the 28 bytes forming 8 keys (N = 8).

4. Combine trios of the 28 bytes forming 12 keys (bytes from the frame length or type are used multiple times) (N = 12).

5. Combine the destination address bytes, source address bytes and frame length bytes into 3 keys. Add 31, 32, or 33 (M_1 = 31, M_2 = 32, M_3 = 33) to each key-data XOR result.

6. XOR the first byte of the destination address with the last byte of the source address forming a single key. Add a value (M = data-byte-position mod 10) to each key-data XOR result.

7. Use the hexadecimal value of byte 12 of the source address to determine key_1. Use the hexadecimal value of byte 12 of the destination address to determine key_2. Alternate the use of the 2 keys and add 2 to every second key-data XOR result and 5 to every fifth key-data XOR result.

8. Generate a key from the source or destination address by adding the 12 bytes. XOR the key with the first data byte. Use the key-data XOR result as the next key. Continue until all data bytes have been processed.

9. Generate a key by adding the 28 bytes. Swap the Nth and (Nth + 2) bytes of the key-data XOR results to mask runs of identical characters.

The single block cipher algorithm also involved the generation of a key schedule from the 28 bytes forming the source and destination addresses and the frame length. Like the stream cipher algorithms, encryption was accomplished by XORing the key schedule with the data segment. However, this algorithm also made use of byte swapping to mask runs of identical characters (a block cipher). Figure 3 shows the basic algorithm. Following Figure 3 is a brief synopsis of the one acceptable algorithm.

```
i = 0;
j = 0;
while (i < text_length) {
    text[i] = (key[j] XOR text[i]);
    i++;
    j++;
    if (j >= N) j = 0;
}
i = 0;
k = text_length;
while (i < text_length / 2) {
    swap ( text[i], text[k]);
    i+=2;
    k-=2;
}
```

Figure 3. Simple XOR block cipher algorithm.

List of Algorithms

1. Form 2 keys by combining the source and destination addresses into 2 integer values. Add the absolute difference of these keys to each data byte. Mask runs of identical characters by swapping the Nth and (last-Nth) characters. Where N = 0, 2, 5, ...

Survey Results

As part of the take-home final exam, the students (49 total) were required to take a 7 question anonymous survey. Each question had 5 possible answers. Answers ranged from 1 (indicating a "no" or "not very") to 5 (indicating a "yes" or "very"). Students were instructed to select the most appropriate answer for each question. Most of the questions were designed to solicit an opinion on a specific question/idea. However, question 2 was vague on purpose and was intended to determine if the students felt that they had learned anything at all. The questions and summary of answers are shown in Table 1.

Table 1. Survey results.

Question	**Max**	**Min**	**Ave**	**Std**
Did you enjoy this research?	5.00	2.00	4.12	0.83
Did you learn anything new by doing this research?	5.00	2.00	4.31	0.77
How difficult was the research?	5.00	1.00	3.61	1.00
Would you like to do another research project like this again (in another class)?	5.00	2.00	4.00	0.91
How successful / appropriate do you feel your design was?	5.00	1.00	3.82	1.07
The assignment was designed to give you a feel for the type of work you may do as a computer scientist. Did the assignment change your perception of computer science in any way?	5.00	1.00	2.98	1.22
Would your recommend that a similar research project be undertaken again in another CS-173 class?	5.00	2.00	4.27	0.84

Results of the survey suggest that the students enjoyed the assignment, felt it was a worthwhile exercise and recommended that a similar exercise be undertaken in future classes. The results from question 2 suggest that many students felt they had learned something. Including the possibility that they did not like computer science! Which was the case for at least one of the students. From question 6, it appears that a significant number of students did not have a clear idea of what being a computer scientist entails[1]. Finally, discussions with individual students also indicated that those who had taken programming classes in high school generally found the task easier, more interesting and had worked towards developing more elegant solutions.

[1] The author's experience with freshman computer science students tends to agree with this result.

Conclusion

Freshman level computer science students were assigned the task of designing and implementing an algorithm for encrypting ethernet packets. Students were allowed to pursue any encryption method they desired including variations of three encryption methods covered in class (block, stream and public-key ciphers). Several assumptions were made to make the task tractable for freshmen level students. However, specific requirements were also made which forced the students into using some of the more complex C++ programming constructs (such as pointers and dynamic arrays). The requirements also forced the students to develop a realistic encryption algorithm.

As a take-home final, students were required to document their design, test cases used and the results of applying the test cases. Students were also required to complete a 7 question anonymous survey. Survey results suggest that the students enjoyed the project and that they would like to do something similar in other classes. Survey results also suggest that, at least prior to the project/class, many students were not sure of what computer science entails.

It is the author's opinion that approximately 25% of the students developed very good solutions. Approximately 65% of the students developed mediocre solutions. Approximately 10% of the students developed unacceptable solutions (in some cases they essentially copied code given out by the instructor). 100% of the programs compiled and ran. These percentages agree with the author's previous experience with programming projects assigned to other freshman level computer science students. However, in all cases, the documentation was little more than a narrative of the program's menu options. To some extent, this was expected as none of the students had yet taken any courses in software engineering.

Acknowledgements

I would like to acknowledge the students whose work was reviewed in this document. They are: Neil Fasteen, Eric Mislivec, Kyle Stern, Tom Simmer, Rong Qu, Patrick Inman, Daniel Lucent, Chad Larson, Anne Erickson, Wade Baird, David Voecks, Chris Peterson, Andy Tidball, Brian Asker and Troy Dahnert.

References

1. Coopers & Lybrand L.L.P., "Microsoft Windows NT Server: Security Features and Future Direction," Information Technology Security Services, 1997.

2. Goldberg, L., "New Encryption Strategy Uses Hardware and Software to Protect Data on Public Networks," Electronic Design, March 6, 1995.

3. March Information Systems - Security Manager Windows NT Security Knowledge Base.

4. Rainbow Technologies - CryptoSwift.

5. Paralon - PathKey Fortress, Rainbow Technologies - CryptoSwiftEN and SignalGuard - SecurNET HSP.

6. http://www.distributed.net/.

7. Schneier, B., "The Twofish Encryption Algorithm: The Current State of the DES,", Dr. Dobbs Journal, December, 1998.

8. Hamzah, K., "http://www.ee.mtu.edu/courses/ee465/groupa/problems.html".