Session Number 1520

# BO (Buffer Overflow): Bad for Everyone

**Kathleen M. Kaplan, D.Sc., Colleen Duran, M.B.A., Lt Col John J. Kaplan (Ph.D., J.D.) USAF**

**Howard University/Duran Consulting/USAF**

## Abstract

No one wants BO, but unfortunately, software engineers have been affecting others with it since programming began. This BO is not the kind discussed in social circles, but it should be; this paper discusses the most offensive BO: Buffer Overflow.

Buffer overflow is the primary offensive tactic in many computer viruses and worms. For example, the Internet Morris Worm of November 1988 would not have been possible without the buffer overflow error in the *finger* command of the UNIX-based computer system. By not specifying a maximum buffer length, programmers had allowed this worm to fill the read buffer and overflow into memory until it had overwritten the return address in the stack buffer. But that was 1988, ancient history in the computer age, surely this could not happen today! Unfortunately, this is not the case. Recently, a buffer overflow was found to be the culprit in the Code Red II Worm; a buffer overflow in the indexing service used by specific Microsoft versions running on particular Windows platforms allowed remote trespassers to execute code on compromised machines.

Buffer overflow can be used by malicious intruders, but it can also cause errors without dishonest intention. The buffer overflow found in the Therac-25, a software-controlled radiation-therapy machine, caused the deaths of three patients and severely injured many more. This overflow was not affected by outsiders, but rather it was a simple programming error; a flag variable was stored in a byte and incremented. The software engineers did not consider the case of incrementing the variable the 256th time, which, too large for the byte, set the flag to zero, an indication that the device was ready.

It seems to be a simple check, "Will my result fit in the allotted space?" Why do software engineers ignore this question? The result from overlooking buffer overflow has led to costly errors, including the loss of human life. Yet, few programmers, let alone software engineers, are aware of the problem. All students who take a programming course must be exposed to the dangers of buffer overflow; only then will this programming error be eradicated.

This paper will discuss the buffer overflow problems in the history of software engineering, the current state of buffer overflow problems, and ways to combat buffer overflow problems in the future.

## 1. Introduction

BO has been a stigma placed upon engineers; it goes with the stereotype of an un-washed engineer hunched over an apparatus for days. This type of BO is not detrimental to most people; it only affects the engineer and nearby workers. Unfortunately, engineers have been affecting more than just those in close contact with another type of BO – Buffer Overflow. Buffer Overflow vulnerabilities are the primary offensive attack targets in many computer viruses and worms. Educators must address buffer overflow in all of their software engineering courses, at the very least, and help to clean up this type of BO – Buffer Overflow.

This paper gives an overview of buffer overflow, including five case studies which have resulted in financial and other losses; in one case, buffer overflow caused the loss of human life. The paper also gives the current state of buffer overflow problems and concludes with some ways to combat the buffer overflow problem.

## 2. Buffer Overflow

Buffer overflow is a simple concept; it is created when an allotted space is too small for the information required. An analogous real-life situation is when an eight-ounce glass is being filled with milk. If more than eight-ounces of milk are poured into the glass, it overflows. In this example, the eight-ounce glass is comparable to the buffer, and the milk is the information. Adding more than eight-ounces to the eight-ounce glass causes the milk to pour over the edge of the glass onto the table or whatever is holding the glass. This is sometimes the case in computers; sometimes the data overflows into the surrounding area of the buffer. In other cases, the data does not intrude into the outside buffer area, but instead the buffer holds results that are not desirable.

While spilling milk is an irritating but innocuous event, buffer overflow may not be. The first case mentioned above, wherein the buffer overflows onto the surrounding area, is what the Morris Internet Worm exploited. The second case, wherein the overflowed buffer did not affect the surrounding area but instead held an undesirable result, caused the loss of human life in the infamous Therac-25 case. These are not isolated situations, but rather just two examples of buffer overflow.

It is interesting that the spilt milk concept and buffer overflow are both easy situations to solve. Preventing spilt milk requires diligence in paying attention to detail, which is the same in avoiding buffer overflow. The computer programmer must be diligent in the proper use of buffers.

The worst-case results of buffer overflow have caused millions of dollars of damage, personal injury, and death. Therefore, every student engineer must be aware of the dangers of buffer overflow.

## 3. Buffer Overflow Problems in the History of Software Engineering

Buffer overflow is a type of computer "bug." Grace Hopper, an author of the computer language Cobol, gave a great discussion of the origin of the term computer "bug" in a presentation that the authors were fortunate to attend. She recalled that the original computers were huge, consisting of thousands of vacuum tubes. These tubes made the rooms in which the computers were housed very hot. So the computer operators would open the doors. Bugs, entering through the open doors, would be attracted to the light in the vacuum tubes and sometimes get stuck in the inter-workings of the computer, causing a shut-down. Therefore, the computer had a "bug" in it, literally! Thus, computer bugs have been around as long as computers have. But when did buffer overflow make its first appearance? It has probably been around since the first software engineer wrote a program using buffers.

The first noted problems with buffer overflow occurred around 1973 [5], but only within software engineering circles. A famous article appeared in IEEE regarding the Therac-25 incidents mentioned above, but when the events occurred in the late 1980's not much attention was focused on buffer overflow. The buffer overflow problem became known world-wide in 1988 with the Morris Internet Worm. It has continued to be a problem since, as evident by the CERT Coordination Center's November 11, 2003 buffer overflow advisory regarding Window's Workstation Service. Below are some infamous buffer overflow problems in the history of software engineering.

### *3.1. Case 1. Buffer Overflow in Health Care, April 1986*

One of the first deaths attributed to buffer overflow occurred in April 1986 [11]; Verdon Kidd received massive doses of radiation from the Therac-25, a software controlled radiation-therapy machine used for cancer treatment, and died shortly thereafter.

The Therac-25 had many computer vulnerabilities; one technician stated that the machine would give up to four error messages a day [11]. Yet, the one that killed Kidd and two other patients was a buffer overflow error.

To ensure safety, the Therac-25 would perform a Set-Up Test after a technician entered treatment parameters, prior to engagement. This Set-Up Test would run through a list of procedures hundreds of times while setting up for one treatment. Within this Set-Up Test, a flag register of eight-bits would indicate problems. If the flag register value was zero, then no errors were present and treatment could begin. A non-zero value in the flag register would indicate that problems remained and treatment could not begin. What the programmer of the Therac-25 did not consider was error number 256. An eight-bit register can hold values from 0 to 255. If 255 errors occurred during the Set-Up Test and then one more, the register value would be 0, and indication that the machine was ready and treatment could begin [1].

This programming error, an error that any freshman engineering student should be able to identify, caused the deaths of at least three patients. The Therac-25 was designed and developed by AECL Medical, a division of Atomic Energy of Canada Ltd, by one programmer who has not been identified [11]. Whether or not the programmer had any formal training is unknown.

### *3.2. Case 2. Buffer Overflow in Internet, November 2, 1988*

The beginning of the world's acknowledgement of buffer overflow as a problem occurred with the infamous Morris Internet Worm of November 2, 1988. Created by a twenty-three year old Cornell doctoral student, Robert Tappan Morris, this small (ninety-nine line, not including object files) program infected over 6,000 computers, causing a virtual shut-down of the internet. The worm was quick; a computer would be unusable in less than ninety minutes from the time of infection [12].

Interestingly, the Morris Internet Worm exploited a buffer overflow problem to carry out its mission. The worm used the TCP *finger* service to gain entry to a system. The *finger* command gives information about a user on a host, such as the user's full name, office location, etc. The Berkeley version of *finger* is simple: (1) a request is read from the originating host; (2) a local *finger* program is run using the request as an argument; and (3) the output is returned to the originating host. This seems reasonable but *finger* reads with *gets()*, another command, which did not check for buffer overflow. The buffer size for *gets()* was 512 bytes; the Morris Internet Worm supplied the *finger* server with a request that was 536 bytes long. This difference of 24 bytes overflowed into the server's stack frame. Thus, instead of executing a system pointer at the end of the 512 bytes, which would return to the calling program, the system pointer was overwritten by the extra 24 bytes in the request. This 24 bytes of code instructed that the request be executed [7]. The rest is history.

At the time, the Morris Internet Worm affected over 6,000 computers. This may seem like a small amount in today's world, but remember that the event occurred in 1988, before the world wide web. No physical damage was caused by the worm, but it was estimated by the United States General Accounting Office that the financial loss was between $100,000 and $10,000,000 [12]. In today's world, the internet is considered a Critical Infrastructure, and if such a worm were to hit, the event would be a national disaster.

### *3.3. Case 3. Buffer Overflow in rlogin, 1998*

A buffer overflow problem occurred in the remote login (rlogin) command and was brought to light around 1998 [10]. This buffer overflow problem is akin to the spilt milk analogy, yet has more sinister complications.

The UNIX command rlogin allows remote login from a local host to a remote host. In doing this, various operations in set-up are required. One requirement is that information about the local host is needed. This information is stored in a TERM environment variable in the remote system's memory. Yet, this TERM is copied without regard to length. A savvy intruder can

modify the TERM information and make it whatever the intruder desires. This modified TERM environment variable can overflow the TERM buffer and cause a buffer overflow. The result of this buffer overflow is that information in the TERM variable can spill into important parts of the remote terminal's memory. This spillage can contain procedures to transfer control from the remote host to a specific address in the local host's memory, thereby allowing an intruder access to the remote host machine.

### *3.4. Case 4. Buffer Overflow in WWW, April 2002*

Buffer overflow occurs via the world wide web, as evident by the Microsoft Internet Information Server (IIS) problem published in the CERT Vulnerability Note VU#669779 [3].

A buffer overflow vulnerability in specific versions of IIS, may allow an attacker to gain upper-level privileges and access to the entire system.

In this software, a user request for a web page is properly processed by including the file into an ASP script and processing it. Prior to processing this request, IIS performs an operation on the user-specified file name, designed to ensure that the file name is valid and sized appropriately to fit in a static buffer. However, in some cases it could be possible to provide a bogus, extremely long file name in a way that would pass the safety check, thereby resulting in a buffer overflow. This could cause code to run on the server.

According to Microsoft, this vulnerability would have to occur as follows: "an attacker who was able to lure a user into clicking a link on his web site could relay a request containing a script to a third-party web site running IIS, thereby causing the third-party site's response (still including the script) to be sent to the user. The script would then render using the security settings of the third-party site rather than the attacker's" [8].

According to Microsoft itself, there are two types of buffer overflow occurring: (1) a wrong-sized buffer is allocated and; (2) no limits are placed on the size of input. As a result, it would be possible for a client to send input that would overwrite most or all of the memory on the system. In doing so, the attacker could act as a legitimate user of the system, perhaps even as the root user.

### *3.5. Case 5. Buffer Overflow in Password Authentication, December 2003*

A recent buffer overflow problem has occurred in the Cisco Application and Content Networking Software (ACNS). This problem is the main focus of the CERT Vulnerability Note VU#352462, posted in December 2003 [2]. This buffer overflow causes an authentication problem.

The password buffer of this software is designed for typed characters of fixed size. By supplying an overly long password, it is possible to trigger a buffer overflow in the authentication module. If a user types more characters than the buffer holds, the overflow causes the operating system to bypass password comparison, which results the same as if a correct authentication had been

supplied. This buffer overflow problem may enable an attacker to execute arbitrary code on the affected device or cause denial of service [9] [2].

This buffer overflow problem is inexcusable in today's software-savvy world. Every engineering student should be exposed to length verification for input, yet, obviously a few at Cisco missed the class.

## 4. Present State of Buffer Overflow Problems

As indicated by Case 5 above, the world is not safe from buffer overflow problems; this problem is current and occurring even today! In twenty years, or more, software engineers have not mastered the basic principle of checking buffer bounds.

A quick search of the world wide web using the term "buffer overflow" on google.com yielded 1,020,000 hits. Of course, this does not show the vulnerabilities, but merely references to buffer overflow.

More reliable search results can be found at the CERT website, cert.org. Using the term "buffer overflow" on the CERT web-site yielded the results in Table 1.

Table 1. Buffer Overflow Information from CERT

| **Type of Search** | **Number of Results** |
|---|---|
| Advisories | 95 |
| Incident Notes | 8 |
| Research | 4 |
| Security Improvement Modules | 2 |
| Tech Tips | 2 |
| Training and Education | 1 |
| Vulnerability Notes | 510 |
| Other CERT Docs | 199 |
| **TOTAL:** | **821** |

The information in Table 1 does not show the current status yet another search in the CERT web-site yielded this information. The CERT's Vulnerability Notes Database was searched using the term "buffer overflow" [4]. This database yielded 321 vulnerabilities, not the 510 of the prior search. Yet, in this search, the dates that the vulnerabilities went public are obtainable. This gives a more accurate view of the current buffer overflow problems. This data is in Table 2.

Table 2. Buffer Overflow Found in Vulnerability Notes by Year

| Year | Number of Results |
|---|---|
| 1997 | 4 |
| 1998 | 2 |
| 1999 | 9 |
| 2000 | 19 |
| 2001 | 79 |
| 2002 | 105 |
| 2003 | 83 |
| **TOTAL:** | **321** |

As can be seen in Table 2, buffer overflow vulnerabilities have not decreased since the CERT began accumulating data. Last year alone the CERT recorded 83 buffer overflow vulnerabilities. The information from Table 1 and Table 2 show between 321 and 510 vulnerability notes and 95 advisories. These are daunting amounts.

On further investigation of the data, perhaps Table 1 shows the main problem. Out of 821 hits, only one was for "Training and Education."

## 5. Combating the Buffer Overflow Problem

Software engineering must take buffer overflow seriously. It is not enough to rely upon previous code, the language constraints, the operating system, or any other measure to protect against buffer overflow. Software engineers must be taught to code for the buffer overflow problem, and at the very least, to identify buffer overflow vulnerabilities.

Thus, education is the key to solving buffer overflow problems. Discussions addressing buffer overflow must be presented to the software engineers of tomorrow, and assignments must be given so that tomorrow's software engineers, our software engineering students, can fully realize the problem and how they will address it. At the very least, every programming assignment should discuss the possibility of a buffer overflow problem. Above that, written, non-programming assignments should be given so that the student can fully realize the scope of the problem.

Following are suggestions of ways in which potential buffer overflow vulnerabilities can be addressed in programming assignments and written, non-programming, assignments.

### *5.1. Tackling Buffer Overflow in Programming Assignments*

In programming assignments, there can be many places where buffer overflow may occur: program input; program data structures; and program output. In each of these, potential buffer overflow vulnerabilities must be addressed.

5.1.1. Program Input

If input is needed, the instructor must discuss an example of a buffer overflow problem regarding input, such as in Case 2's discussion of the Morris Internet Worm above. A question can be asked of the engineering students such as, "How could this have been prevented?" The students will give suggestions, then the instructor can follow up with, "How will you prevent input buffer overflow in your program?" The students can also be required to answer this question in their program comments or supporting documentation.

5.1.2. Program Data Structures

Within the program itself, there may be arrays, linked lists, or other data structures which may cause buffer overflow problems. The students can be taught these situations through an example such as Case 5, where the length of the string buffer was not checked. Again, the students should be prompted to think for themselves: "What could have been done?" Then this too should be followed up with "What will you do?" and supported by the student's documentation.

If registers are needed, a discussion of the Therac-25, also given above, or a similar case, must be discussed. The software engineering students must be constantly reminded of the dangers associated with buffer overflow.

5.1.3. Program Output

Output should also be addressed. The students should be aware that in today's world, a software engineer rarely acts alone. His or her program will most likely be incorporated into another program. Thus, the output written may be input to another program. If output is allowed to overflow the intended amount, then that may cause a buffer overflow vulnerability in the end result. The cases given above can be used to address this as well. "What if the rlogin command in Case 3 is initiated through output of a program? Who is responsible for buffer overflow checking?" Let the students respond, so that the issue can be fully realized: buffer overflow is every software engineer's responsibility. As in the input and program itself, the student's solution to buffer overflow checking should be required in the documentation.

*5.2. Suggestions for Written Assignments*

Not all courses require programming; many include written assignments. One assignment in every course, where buffer overflow could be a concern, should address buffer overflow vulnerabilities.

5.2.1. Case Review and Presentation

A good way for the students to identify the problem is to read some literature and to present a paper on the subject. For example, the case studies in this paper could be assigned to students in groups. The groups could research the problem in depth: initial flawed design, vulnerability attack/problem, result. Each group could then present the findings to the class.

5.2.2. Cert review, Synopsis, and Student Solutions

Another interesting assignment would be to have the student search the CERT web-site [4] for the latest buffer overflow vulnerability and write a short synopsis of the problem. This could be followed by ways in which the student would suggest to solve the problem and to prevent future problems.

5.2.3. Patent Review and Paper

The student could search the United States Patent and Trademark Office web-site [13] using "buffer overflow" as the search term. One of the patents of the search result could be read and reviewed. The student could then be instructed to write a paper on this patent describing the methods used to address buffer overflow.

5.2.4. Buffer Overflow in TCP/IP

Students enjoy learning concepts about what they use daily. There are many buffer overflow problems in TCP/IP, not just in Case 3 given above. Another is found in the IP Messenger; a buffer overflow occurs when a user attempts to save a file containing a long filename. There are many more and students could be asked to find all the TCP/IP buffer overflow problems they can. One of the authors teaches Data Communication Networks, and this assignment is very popular!

In this section of the paper, the discussion of buffer overflow vulnerabilities to students was presented. Ways in which to incorporate the problem of buffer overflow into the education of software engineering students were presented: both in programming assignment and written assignment suggestions. Education is the only method that can be employed to combat the buffer overflow problem.

**Conclusion**

Buffer overflow continues to be the primary offensive tactic in many computer viruses and worms. One does not have to reach back in time to the 1980's the Morris Internet Worm example to find a buffer overflow problem. A very recent example occurred in December 2003 regarding Cisco's password authentication. These two examples were given in this paper along with two others, regarding rlogin and Microsoft's Internet Information Server (IIS), which show that intruders can use buffer overflow to obtain access to other systems.

This paper has also shown one example, the Therac-25, which did not need an intruder to activate buffer overflow for malicious intent; the machine did so of its own accord. As stated above, a buffer overflow problem caused the deaths of at least three patients.

The paper also showed the current state of buffer overflow by reviewing the CERT website. Since CERT has been collecting data, there have been between 321 and 510 vulnerability notes

and 95 advisories pertaining to buffer overflow. Note that 83 of the vulnerability notes were released to the public in 2003.

The simple check, "Will my result fit in the allotted space?" is not being addressed by all programmers. The result from ignoring buffer overflow has led to costly errors, including the loss of human life. Education is the key to eradicating the problem of buffer overflow.

The paper also focused on possible ways to educate software engineering students. Programming assignments must also include discussions on buffer overflow. These discussions must focus on all aspects of the assignment: input, program itself, and output. A student should be allowed to think the problem through: "How could the problem be solved and how did I solve it?" The student should also be required to give this answer in the comments and/or other supporting documentation. Non-programming assignment suggestions were given too.

Software engineers should not be remembered for their BO (Buffer Overflow) problems. BO was a problem for software engineers in the 1980's and is still a problem today. The only way that software engineering will eradicate BO is to educate software engineering students so that it's cleaned up at its source.

**References**

[1] Baase, Sara, *A Gift of Fire: Social, Legal, and Ethical Issues for Computers and the Internet*, 2nd ed., Prentice Hall, 2003.

[2] CERT Coordination Center, "Cisco ACNS contains buffer overflow vulnerability in the authentication module when supplied an overly long password," *CERT*, Vulnerability Note VU#352462, Dec. 12, 2003, accessed January 2004.

[3] CERT Coordination Center, "Microsoft Internet Information Server (IIS) 4.0, 5.0, and 5.1 buffer overflow in chunked encoding transfer mechanism for ASP," *CERT*, Vulnerability Note VU#669779, Apr. 10, 2002, accessed January 2004.

[4] CERT Coordination Center, Vulnerability Notes Database, http://www.kb.cert.org/vuls/, accessed Jan. 2004.

[5] Donaldson, Mark E., "Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention*," SANS Institute*, 2002.

[6] Levenson, Nancy G., and Clark S. Turner, "An Investigation of the Therac-25 Accidents," *IEEE Computer*, July 1993.

[7] Litterio, Francis, "The Internet Worm of 1988," http://world.std.com/~franl/worm.html, accessed January 2004.

[8] Microsoft, "Microsoft Security Bulletin MS02-018, Cumulative Patch for Internet Information Services (Q319733)," *Microsoft*, April 10, 2002. http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms02-018.asp, accessed Jan. 2004.

[9] Pfleeger, Charles P. and Shari Lawrence Pfleeger, *Security in Computing*, 3rd ed., Prentice Hall, 2003.

[10] Rogers, Lawrence R., "rlogin(1): The Untold Story," *Carnegie Mellon University*, Software Engineering Institute, CMU/SEI-98-TR-017, Technical Report, Nov. 1998. http://www.cert.org/archive/pdf/98tr017.pdf, accessed Jan. 2004.

[11] Rose, Barbara, "Fatal Dose: Radiation Deaths linked to AECL Computer Errors," *Saturday Night*, June 1994. http://www.ccnr.org/fatal_dose.html, accessed Jan. 2004.

[12] Schmidt, Charles, and Tom Darby, "The What, Why, and How of the 1988 Internet Worm," http://www.snowplow.org/tom/worm/worm.html, accessed January 2004.

[13] USPTO, United States Patent and Trademark Office, http://www.uspto.gov.

[14] Wikipedia, "Buffer overflow," *Wikipedia, the free encyclopedia*, http://en2.wikipedia.org/wiki/Buffer_overflow, accessed January 2004.

## Biographical Information

KATHLEEN M. KAPLAN, D.Sc.
Dr. Kaplan is an Assistant Professor in the Department of Systems & Computer Science at Howard University. She is also a Registered Patent Agent licensed to practice before the United States Patent and Trademark Office. She can be reached at kkaplan@howard.edu.

COLLEEN M. DURAN, M.B.A.
Ms. Duran has an extensive background in business and elementary education. She has recently focused in the banking and stock management industry. Currently Ms. Duran is a business and educational consultant and can be reached at cmbduran@aol.com.

JOHN J. KAPLAN, Ph.D., J.D.
Dr. Kaplan is a Lieutenant Colonel (Lt Col) in the United States Air Force and a Patent Attorney. Lt Col Kaplan is currently the Commander of the 694th Support Squadron. He can be reached at 694spts.cc@ft-meade.af.mil.