Buffer Overflows: Implications for Civilian Cybersecurity

Andrew Hoskins
ahoskins@microsoft.com

Mark O. Ihimoyan
mark.ihimoyan@microsoft.com

Adeel Rasul Iqbal
aiqbal@uclink.berkeley.edu

Keunwoo Lee
klee@cs.washington.edu

Thanh Nhut Nguyen
ttnguyen@uclink.berkeley.edu

October 24, 2005

Abstract

A *buffer overflow* is a security problem wherein a computer program defect permits an adversary to gain unauthorized privileges on a target machine. We describe an experiment in which we attack a buffer overflow defect in a sample program. Based on this experiment and other research, we analyze the difficulty this class of attack, and describe implications, including anticipated economic damages and impact on terrorist objectives. Finally, we discuss defenses and public policy implications.

1 Attack Analysis [Nguyen]

In this section, we describe the experiments performed by our team to investigate buffer overflows. Then we summarize results, including the difficulty, exposed vulnerabilities, and a summary of possible defenses.

For clarity, we first define some terms. An *exploit* is a program which permits a user to perform an action which (s)he is not authorized to perform. An *exploiter* is a user who uses an exploit. A *vulnerability* is a flaw in a program which permits an exploit to occur. We distinguish an exploit from an *attack*, which we define as an attempt to discover a vulnerability and build an exploit. We call the entity who performs attacks an *attacker*. A *buffer* is a parcel of memory where a running program temporarily stores data. The *stack* is region of program memory where information related to current and future computations is kept; a buffer may be stored in the stack. A *pointer* is a a piece of data that refers to another piece of data or code — a pointer names a location in memory, in much the same way that a postal address names a location in the physical world.

1.1 Description of Vulnerability

At a high level, in a buffer overflow vulnerability, a specially crafted input is sent to a program, and improper handling of this input causes the program to perform actions on the behalf of an exploiter.

A buffer overflow exploits is made possible by the fact that programs commonly write data into a buffer without checking if data fits into this buffer.

To better understand this concept, the visual image of writing on a series of crumpled pieces of paper may help. Imagine an individual sitting at a desk and writing a long sentence off a notepad onto one of dozens of pieces of crumpled paper surrounding this notepad. If the writer's memory wanders, he or she may fail to notice that the sentence has gone from the end of one piece of paper to the next. Ultimately, a human will catch him/herself. But computer programs are oblivious to such an error. After a buffer in the stack overflows, potentially any information in the stack can be overwritten with information provided by the exploiter.

One of the pieces of the stack is a pointer to code that will be executed in the future. By overwriting this pointer, the exploit tricks the program into executing code provided by the exploiter.

1.2 Methodology: Description of Attacks Attempted

Three pieces of information are required to craft an exploit. First, it must be known *where and when a program expects input* that's copied into a buffer. Second, *the size of the buffer* to be overflowed must be known. Third, the *address of the buffer* in memory must be known. The purpose of an attack is to discover these three pieces of information, and then craft a program which uses this information to craft an exploit.

The first piece of information was provided by the designers of the exercise. The technical members of our team attempted three separate attacks to discover the second and third pieces of information: *source code inspection*, *automated brute-force trial-and-error*, and *controlled inspection of the program execution* (debugging). In the first of these attacks, we permitted the simulated attacker access to the *source code* of the target program.[1] In the second and third of these attacks, we permitted the simulated attacker access only to a copy of the executable program, without source code.

Because the target program was executed *setuid root* — a technical term meaning that it operates with administrative-level privileges — successful attacks granted administrative privileges on the target machine.

1.3 Results: Estimated Difficulty, and Vulnerabilities

Estimated Difficulty Red Team 7 successfully performed two classes of attacks: the open-source attack, and the brute-force attack. The open source attack was successfully performed independently by two team mebers, whereas the brute-force attack was successfully performed by one member. The debugging attack did not succeed. The total effort expended included about 1-2 hours of effort common to both attacks, and 1-2 hours specific to each attack. Given the information provided (commonly available on the Internet), either or both of these attacks could be devised by a moderately skillful programmer in about 3-6 hours. Details follow.

Open-source attack. This attack relied on access to the target's source code. After examining the code, the team modified it to output a piece of key information, and ran this modified executable. By doing this, our team obtained the three pieces of information necessary for attacking the main target.

Brute-force attack. This attack relied on having the ability to run the target program repeatedly — on the order of tens of thousands of times — in an attempt to find the weakness. The team wrote an automated program that tried many variations of the general attack, and ran it. When the script concluded, the team had enough information to complete the attack.

Debugging attack. This attack relied on having the ability to run the program in a common programming tool called a *debugger*. This attack did not succeed, due to technical properties of the debugger.[2]

Vulnerabilities Exposed The attack gave the team access to the machine's "root" superuser account, which has privileges to execute any program. This allowed the team to read, change, or observe any activity, program or resource on the machine.

The attacker would be able to read files on local disks, files on any shared network disks accessible to users on the machine, all traffic on the local network that's connected to the machine and anything that can be picked up by input devices (e.g., microphones, keyboards, cameras) connected to the machine. If the machine were to be used for e-mail, the team would be able to view these messages. If the user were to use this machine to log onto another resource, the attacker could log onto that resource too.

Furthermore, the attacker can use this machine to anonymously launch further attacks, or wipe and destroy the hard disk or other resources connected to the machine. A clever attacker may be able to observe users logging in, and thereby obtain passwords for those users. Once such passwords are obtained, they can be used to access other resources on the network.

A real attacker would most likely refrain from destroying the hardware in practice, because once the team has access to the resource, it would be more useful to maintain the system and use it surreptitiously.

1.4 Summary of Defenses

The successful attacks could have been prevented, or made less likely, in several ways.

- The source code could have been kept secret, defeating the open source attack.
- The buffer overrun vulnerability could have been found and removed in the program. The programmer could have relied on a more secure library to do its work, or the programmer could have been smarter. The programmer could also have used another, safer programming language.

[1] Source code is the representation of the program that programmers commonly use when writing software. In general, proprietary software companies do not distribute their source code, but so-called "open source" software organizations do.

[2] Loading a program into a debugger perturbs the address space of the process. This defeats efforts to determine the address of a buffer when the program is run outside a debugger.

- The program could have executed without administrative privileges. Thus, even with a vulnerability, an exploiter, possessing fewer privileges, could do less damage.
- Address-space randomization in the installed operating system would have caused characteristics of the program change with every execution, thus making it more difficult to prepare and launch an attack.
- The compiler could have inserted buffer overrun checking into the program, such that once an overrun was detected, it would crash rather than execute user code.
- The program could have made the ”stack” memory space non-executable, preventing the user from executing the data he inserted into the program.
- The program or underlying system could have prevented the user from executing the program multiple times, preventing the brute-force attack.

We examine the broader impact and cost of some of these defenses in more detail later, in Section 3.

2 Details: Vulnerability Assessment

2.1 Estimated Potential Damage: Case Studies [Iqbal]

Putting a dollar value on cyberattacks can be correlated with the phrase, “taking a shot in the dark.” There is no way to accurately judge how significant an impact an attack or exploit, similar to the one conducted by Red Team 7, can have. Nicholas Weaver, Computer Security researcher at UC Berkeley’s International Computer Science Institute, states [13]:

> This is very hard to do. Some computers have value $0; e.g. a home machine, where it gets reformatted instead of mowing the lawn this weekend. Yet if that same machine is used to access a bank account, and the account got compromised, then the loss may be many thousands.

Weaver adds that time of impact is critical for the determination of financial impact:

> Even trying to compute average damage figures is nigh impossible: How do you account for lost time? There is a huge nonlinearity, even in a relatively important business system: Down for 5 minutes? You go to the break room. Down for an hour? So what. Down for a day? That may be a problem. Down for a week? Chapter 11 (bankruptcy).

The answer ultimately lies in the details and specifics of the type of attack, the location of the attack, the number of people directly influenced, etc. “It really comes down to ‘It depends,’ for both damage and response/prevention strategies,” Weaver states.

With this context in mind, below Red Team 7 provides estimates to the best of its abilities on the impacts of a buffer overflow type exploit on three specific test locations.

2.1.1 Case Study: Private Home

It is important to keep in mind that damage made to any location is not only financial, but also psychological. And psychological impacts can result in financial impacts in the long run, e.g. through personal damage lawsuits, the implementation of brand new security systems for appeasement, new regulations, etc.

Psychological impacts may include loss of data of sentimental value such as family photographs, or the loss of “peace of mind”. The value of impacts like these are difficult to quantify economically, as the value is subjective, but may range from zero to millions of dollars depending on the individual.

Direct financial impacts come in two forms: loss of data integrity, and loss of data confidentiality. Losses of integrity may include destruction of work, which could result in severe impacts on an individual’s academic or professional career totaling tens of thousands of dollars for middle class individuals. Losses of confidentiality may include personal account information, credit files, tax reports, bank statements, legal documents and passwords. Breaches of confidentiality in this information could lead to identity theft, with damages again totaling in the tens of thousands of dollars for middle-class individuals. In the aggregate, losses such as these could be quite large: individual consumers lost $3.8 billion in 2003 due to identity theft [6].

2.1.2 Case Study: Corporate VP of Ordering Stuff from China

According to Wal-Mart, the company "procures high volume of merchandise from China and exports to the rest of the world through its Global Procurement Center located in Shenzhen." Wal-Mart's direct and indirect procurements have increased annually by $2 billion to $3 billion since 2001, totaling $18 billion in 2004 [11]. The company works with 20,000 suppliers in the country, and has a total of 5,3111 store units globally in ten different countries. Wal-Mart has more than 1.6 million associates worldwide and had $285.2 billion in sales for fiscal year ended January 31.

Any impact to the company distribution would be significant, especially depending on the time necessary to address the impact. Distribution management is a very exact science. Any error or delay has major economic reverberations in the market, especially for the host company. Shipping centers work on a precise schedule. The VP of Trade with China handles close to $20 billion annually. His computer also contains information on exactly who his or her company trades with, and the quantities of those trades. He is also likely connected to the entire company network. If the exploiter is interested in selling confidential data, he or she can. If the exploiter is interested in contacting Chinese suppliers, especially as a representative from another retailer, he or she can. If the exploiter wants to slow the system down, he or she can.

In her statement in May to a House subcommittee, Sandra Thompson, of the Federal Deposit Insurance Corporation adds that "controlling the exposure is dependent upon the time it takes to verify the fraud and discover the source and extent of the compromise." [9] Delay could also bring down the company share price very quickly, lead to negative press reports, and destroy consumer trust in the company. A loss of shoppers leads to a loss of revenue. And, if the problem is long enough, it may raise prices at the "always low price" stores. Trust would also be an issue with the suppliers the company deals with in China. Losing one supplier could lead to many millions in losses.

Jain suggests that total impact of a high-level compromise for a large company could reach the billions. [7]

2.1.3 Case Study: Charles Schwab computer used to place buy/sell orders on NYSE

This computer has more of a direct tie to the market than does Wal-Mart. A Schwab computer trading on the New York Stock Exchange, if hacked into, can be the host for a market turn around.

The company, according to its annual report filings on EDGAR had $1.081 trillion in 7.3 million active client accounts as of December 31, 2004. Damage could be at least in the billions. The stock market is very time sensitive. Any delay or slowdown results in huge losses on a per minute basis. In the long run, so would non-trading revenue, as a lower principle results in lower returns. On a daily basis, the company has an average revenue trade of $156,400. This amount would likely become a negative with any slow down, resulting in million of losses in the long run.

A security breach would also cause clients and employees to lose trust. Net new client assets in 2004 were $50.3 billion, close to 5 percent of overall assets. If the number of new clients plunges, so will the company revenue. Internally, the company trades for its own employees through their 401(k) plans. These individuals will also lose trust in their parent.

2.2 Estimated Attack Value for Terrorist Aims [Lee]

When considering the value of this class of attack to a terrorist organization, several questions arise. First, is the attack *feasible*, i.e. could a terrorist organization realistically perform it? Second, is it *scalable*, i.e. do the results in this experiment extend to serious attacks? Third, what is its *utility*, i.e. how could these exploits based on these attacks be used to accomplish terrorist aims?

2.2.1 Feasibility

To *construct* the attacks we conducted, a basic background in computer science would be required, comparable to that of an advanced undergraduate in computer science, or a bright self-taught programmer. To *execute* a working exploit, all that's required is access to a computer and basic computer literacy skills, such as the ability to use a search engine and to download and install a program.

Both of the above are available to many terrorist organizations. Walter Laqueur notes that contemporary organizations successfully recruit middle-class, educated members [8]. The best-funded organizations have been known to fund "scholarships" for aspiring members to learn chemistry or other sciences; such organizations could easily fund scholarships in computer science. And, of course, computer literacy and access are widely diffused in wealthy First World nations, or even the emerging middle class of developing nations.

Therefore, we conclude the following. First, all but the poorest terrorist organizations possess the ability to execute exploits. Second, better-funded terrorist organizations also have the ability to construct new exploits.

2.2.2 Scalability

The program we attacked was particularly amenable to study; to what extent do these results extend to large, realistic target programs? As previously noted, successful buffer overflow exploit requires answers to three questions. First, where does the program accept input that may allow buffer overflow? Second, what is the size of the buffer to be overflowed? Third, what is the address of this buffer?

In this experiment, the answer to the first of these questions was given to us. In a realistic target program, it might take a great deal of trial and error to discover where a possible buffer overflow exists. Our experiment does not give us sufficient information to quantify time or money costs to do this this with a realistic target program. However, such overflows are discovered by the worldwide hacking community at a rate on the order of hundreds per year, suggesting that it is not difficult.

We successfully answered the second and third questions using two attacks: a source examination attack, and a brute-force attack. We can more easily quantify the scalability issues of these attacks.

Scaling the source examination attack. A large program may require several times more effort for this attack than our test program. However, this effort does not scale in direct proportion to the code size, but somewhat less. Furthermore, the source examination attack took about one programmer-hour or less for both attackers; even assuming a hundredfold increase for a large program, it would only take 100 programmer-hours. Three programmers working for a week would suffice.

Scaling the brute-force attack. The only additional obstacle to performing this brute-force attack on a realistic target might be processing time — a realistic program may do a significant amount of computation, making the brute-force trial take longer. For our target, this attack takes tens of hours. Even if executing the brute-force attack were 100 times slower than for our target, brute-force scan would only take thousands of machine-hours, or on the order of weeks with tens of computers (total cost on the order of $10,000).

In short, these attacks easily scale to realistic target programs, with generous estimates of the additional cost being on the order of $10,000 or a few weeks of human and machine time — resources that are easily at the disposal of many terrorist groups. Furthermore, all of the above estimates apply only to attackers who wish to craft their own exploits, without any assistance from others. In reality, there exists a large worldwide community of attackers who are continually developing exploits and sharing information about them. A terrorist group could invest essentially no money or labor, and simply wait for this community to provide working exploits.

Therefore, we conclude that the vulnerabilities we exposed easily scale to realistic and important targets, including web servers, `ssh`, or other network-connected programs in common use.

2.2.3 Utility

Our attacks grant the attacker total control over some machine. There are several ways such a machine can be valuable — it may provide access to information, computational power or network connectivity, the chance to destroy data, or a platform for mounting attacks on other machines.

The aim of a terrorist organization is to cause violence against one group of people for the purpose of psychological effect on another group. With this in mind, we can envision several uses of machine resources for terrorist ends. We can group these uses into two general categories: *tactical* uses, and *organizational* uses. By tactical uses, we mean the use of computing resources to execute a specific attack. By organizational uses, we mean the use of computing resources to build or to grow the terrorist organization itself.

Tactical uses As Bale and Ackerman have noted [2], violence has the most psychological effect when it is spectacular and acts upon primal fears, such as the fear of contamination (as with biological attacks). Currently, most tactical uses of compromised computing resources seem relatively ineffective for this purpose.

Although much of our nation's *economic* and *utilities* infrastructure relies on computing technology, it is difficult to see how temporarily disabling commercial transactions, or even infrastructure like the electric power grid, would cause dramatic psychological effects. In 2000, abuse by energy traders caused massive brownouts in California, yet this did not cause widespread panic. Therefore, we believe that, at present, compromised IT resources have relatively little "direct" tactical value for terrorist aims.

One might wonder, then, whether there are "indirect" tactical uses. Do compromised computing resources offer significant value for acquiring conventional, chemical, biological, radiological, or nuclear weapons? We believe the answer is no. For each of these classes of weapons, IT resources seem to be of dubious value. Conventional weapons are already widely available to even the least sophisticated terrorist groups. The limiting factor in acquiring chemical weapons is chemistry expertise and access to raw materials, both of which both are already accessible to many terrorist organizations. The limiting factor in acquiring biological weapons or radiological is either physical access to a weapons lab, or the technology to weaponize biological agents. Access to labs is not purely computer-mediated, and weaponization technology is a problem of physical engineering. Finally, the limiting factor in acquiring nuclear weapons is either access to refined uranium (for fission bombs) or the ability to acquire a prefabricated weapon (for fission or fusion bombs). Both of these rely on either state sponsorship, or the ability to subvert human agents of a state, neither of which depends on computers.

We conclude that even indirect tactical uses of compromised IT resources, *for terrorist aims*, are minimal.

Organizational uses We envision two possible organizational uses of compromised resources: for communication, or for financial gain.

First, modern terrorist organizations need IR for the same reasons as any other organization: they must communicate with each other, and they must publish information about themselves in order to communicate with the broader world. However, terrorists face an adversary (law enforcement and intelligence agencies) that attacks legitimately owned resources. Any computing resource that a terrorist organization acquires legally presents an opportunity for investigation. Therefore, compromised machines may be valuable as proxies to disguise the origin of email and other traffic, or as servers to host content such as propaganda videos.

The value of untraceable IT resources is difficult to measure in dollar or hour terms, since it enables terrorists to elude capture, or forces law enforcement to use more expensive investigative techniques. The value of eluding capture varies widely depending on the individual — a fresh recruit may be essentially worthless, whereas a well-trained operative may be critical to the organization.

Second, because compromised resources may contain data with financial value, terrorist organizations may use the information gathered on IT resources for financial gain. This information may be of direct financial value (credit card numbers), or it may enable blackmail (via capture of private data) or control over material resources (via impersonation of authority, as with the Wal-Mart manager described in Section 2.1.2).

Again, the value of all these avenues for financial gain varies widely. However, Kirk Bailey has noted that existing cybercriminal networks have been able to reap many millions of dollars annually from compromised credit card numbers [1]. If this attack were used to compromise a machine running an e-commerce database — for example, a server at an online retailer — then it could enable fundraising on a similar scale.

3 Details: Defenses [Hoskins]

What Market Momentum for Improved Security Is There? Software is easy to distribute once it is created. Also, software engineers and computer scientists are passionate about improving the state of the art. Improving hardware and voracious consumer appetite has supported this rapid advancement. These things have contributed to the large-scale insecurity present in our software infrastructure, but are also the primary reasons why there is considerable momentum to make software secure against hacking.

The software market has become increasingly concerned about security, and people are increasingly willing to pay, as the internet gets larger and more fundamental to profit. Meanwhile, software providers are recognizing how lucrative it is to be able to boast good security, and how damaging it can be when your vulnerabilities are exposed. The effect of a hacking attack is often far more damaging than the specific impact of the attack, as it undermines confidence in the software and inspires copycat attacks until the problem is fixed.

It's true that the best protection against attacks is simply good administration and policy in the companies that distribute and use software. Having quality control and not allowing people to get hold of things like passwords is important, quite aside from technical issues. These costs will be largely incurred above and beyond simply avoiding software vulnerabilities, so here, we focus on technical costs.

Creation of software has a "pay once, benefit many people" or "pay once, benefit for a long time" dynamic. Software that's created today can be pushed over the internet to millions of computers. In addition, today's software provides a basis for future, more complex software, and its design aspects inform future designs, and thus the state of the art improves. The market is thus willing to invest in enhanced security, because there is feeling that we can drastically improve our security, that it's not just an arms race.

So we have two contributing factors: a problem companies feel is soluble, and a market which demands a solution. These factors will induce greatly increased computer security in the coming years. Next we'll discuss some ways in which this is happening, and at the end we will discuss options should market-driven progress fail to be satisfactory.

What Are Some Security Improvements Is The Market Willing To Pay For? There are three ways to avoid being damaged by exploits such as buffer overrun attacks. One is to lower the attack surface: don't expose your program to input from the user in unnecessary ways; for example, don't install potentially risky programs on a server by default. The second is to separate the attack points from critical systems; for example, our target executable for this exercise should have not run under administrative privileges. Or an internet browser could operate within a shell and not be able to affect the larger computer system. The third is to make the software itself free of vulnerabilities, and can be done by hiring talented developers, having an excellent process of quality control, and using good meta-software tools which make your software inherently resistant to exploits recover if a vulnerability is indeed exploited. It can also be achieved to a lesser extent by having an efficient way to provide fixes for broken software.

Successfully pursuing all three of these items involves hiring talented and experienced developers and managers, which incurs perhaps a 10-20% overhead over hiring their less talented peers (assuming roughly $300,000 of annual cost per developer). Such developers can create robust designs, and the creation of the better software is not more expensive beyond the initial cost of the people, who can do more with less time. Also, good design becomes cheaper over time, as the state of the art improves, and as people learn from the mistakes of the past.

The third item is particularly interesting, because its effectiveness doesn't presuppose that people are flawless. So we focus on some items here as fundamental and cost effective ways to achieve security.

In the short term, many companies, most visibly Microsoft and Apple, have developed ways to push patches for broken software out to customers. Millions of customers regularly (perhaps monthly) receive updates that fix vulnerabilities. For some fixes, it is possible to "hotfix" the client software so that its host need not even reboot when it gets the patch. The technology behind hotfixing is improving. The cost of protecting customers from discovered vulnerabilities post-shipment has thus gone from prohibitive to relatively cheap in just a few years. The up-front cost to develop these mechanisms was large, but the recurring cost is small. A server can avoid being reliant on hotfixes by having multiple servers redundantly provide the same service. This is often necessary due to transaction volume anyway, and need not always be considered an additional cost. In addition using patch-related best practices can reduce patching costs. A study showed that centralizing IT operations can reduced patching costs by up to 55 percent. This study also showed that adopting an end-to-end patch management system can reduce costs by up to 44 percent.

The runtime environment software uses, or the construction of the software itself, can be inherently more secure. For example, Microsoft's latest compilers provide buffer-overrun checking which cause the program to crash rather than execute nefarious code when a vulnerability is found. Moreover, when such a thing happens, an error report can be sent to Microsoft, and Microsoft can then find and fix the issue. Recent versions of Linux provide Address Space Layout Randomization, which makes it difficult for hackers to use hardcoded values when making their attacks, and much more difficult to make viruses and worms. New Linux modifications also involve putting more restrictive permissions on segments of executable code, so that it is more difficult for an attacker to write or execute to it. These mechanisms make newer software more secure very cheaply, and either would have protected against our attack for this project.

Also, software can be made type-safe using languages like Java, C++/CLI, Ada, and C#. This prevents users from writing input to segments of the executing program which aren't designated as accepting user input. More recent versions of the C Runtime Library (CRT) contain enhanced security features which make it easy for software developers to eliminate potential points of vulnerability and safe guard against Stack based exploits possible in previous versions of the library. Rewriting old software in type safe languages is quite expensive, but this cost can often be amortized by the fact that the software often needs to be rewritten for other reasons. Writing type-safe code has many other merits; it can actually improve developer productivity, as software bugs tend to be discovered at compile-time rather than run-time. However, not all software can be written in a type-safe language, either because it is impossible (some system code), or because the runtime performance impact of type-safety is often significant for time-critical applications.

All of the things listed here are cost-effective because (a) it is easy to distribute software, (b) it is easy to use protection mechanisms which can benefit all or most software, or (c) because costs of increasing software security are often the same as the costs of increasing software quality.

How Far Will The Market Take Us? But what is the result of these trends? Is it enough? It's clear that the turnaround hasn't been and will not be immediate; there is too much complexity and a good deal of inertia due to currently installed software. Yet trend toward greatly improved security will continue, and certain simple methods like buffer overrun attacks will become a thing of the past within a few more years. Hackers will have to resort to other means.

But keep in mind that most of the market does not demand absolute perfection: people back up their data, they secure data cryptologically, and they can reinstall systems that are compromised. In some cases, the market simply does not understand how much financial damage actually results from an attack.

Perfect security will be reserved for those institutions who are willing to pay dearly, such as banks and governments. Such institutions pay hackers to try to break their systems, and develop in-house software with intense quality controls. Unfortunately, because these organizations develop customized solutions, it's unlikely that total security will be consistently achieved across these critical organizations unless some forces other than the market come into play.

The government has begun investing in education programs as well as mechanisms to make sure that its own systems are secure. This is not enough. In particular, Berkowitz and Hahn think the U.S. government lacks a "mechanism in the policy that holds public officials, business executives, or managers responsible for their performance in ensuring cyber security; nor is there a mechanism that ensures failure will result in significant costs to the responsible parties." A bank that promises its customers that their data is secure must be held accountable if there is a breach.

It is important that the government patch up the legal issues regarding cybersecurity, and protect its own resources. It must be ready to patch holes in what rules the market enforces. However, the market of the 1990s is not the same of the market of today, and it is crucial to recognize that there is considerable momentum in this area now.

References

[1] Kirk Bailey. Lecture, 2005.

[2] Jeffrey Bale, Garry Ackerman. Recommendations on the Development of Methodologies and Attributes for Assessing Terrorist Threats of WMD Terrorism. Technical Report, Center for Nonproliferation Studies, Monterey Institute of International Studies, 2004.

[3] Bruce Berkowitz, Robert W. Hahn. Cybersecurity: Who's Watching The Store? March 2003.

[4] Charles Schwab Corporation, Form 10-K. U.S. Securities and Exchange Commission. 12 Dec 2004. `http://yahoo.brand.edgar-online.com/doctrans/finSys_main.asp?formfilename=0000316709-05-000006`

[5] Computer Security Institute. Cyber crime bleeds U.S. corporations, survey shows. 7 April 2002. `http://www.gocsi.com/press/20020407.jhtml`.

[6] Gregory Gerard, William Hillison, Carl Pacini. What Your Firm Should Know About Identity Theft. The Journal of Corporate Accounting and Finance, p. 1. May/June 2004.

[7] Gaurav Jain. Cyber Terrorism: A Clear and Present Danger to Civilized Society? Information Systems Education Journal. 12 August 2005. `http://isedj.org/3/44/`

[8] Walter Laqueur. No End To War: Terrorism in the Twenty-First Century. The Continuum International Publishing Group, Inc., New York, 2004.

[9] Sandra L. Thompson "Enhancing Data Security." House Subcommittee on Financial Institutions and Consumer Credit. 18 May 2005.

[10] Theo Forbath, Patrick Kalaher, Thomas O'Grady. The Total Cost of Security Patch Management".
http://download.microsoft.com/download/1/7/b/17b54d06-1550-4011-9253-9484f769fe9f/TCO_SPM_Wipro.pdf

[11] Wal-Mart Stores, Inc. An Introduction to Wal-Mart.
http://www.wal-martchina.com/english/walmart/index.htm

[12] Wal-Mart Stores, Inc., Form 10-K. U.S. Securities and Exchange Commission. 31 January 2005.
http://yahoo.brand.edgar-online.com/doctrans/finSys_main.asp?formfilename=0001193125-05-066992

[13] Nicholas Weaver. Personal interview, 21 October 2005.