Software Process

Eric Leonard
University of Washington
leonarde@u.washington.edu

Chad Parry
University of Washington
jchad@u.washington.edu

Daryl Sterling Jr.
UC San Diego
darylsterlingjr@gmail.com

Charistel Ticong
UC Berkeley
charis_t@uclink.berkeley.edu

Introduction

It could be said that one of the reasons that cybersecurity is such an important topic is because the software products that currently exist in the marketplace are so flawed. This could be an overstatement, but it is certain that existing software processes could be improved. Enterprises have long been concerned with developing the most feature-rich applications to give them a competitive advantage in the marketplace. This approach has created vulnerable code that falls prey to the threats of cyberterrorism, hacking and the financial risk posed by privacy legislation and litigation. Development processes must change to improve the security and privacy of software. Recommendations for public policy as well as software practices will be introduced.

Factors Driving the Need for Changes to Modern Software Practices

New Opportunities

In the beginning, there was no internet. There were not 350 million hosts all sharing data and information at blazing speeds. There's no way the four hosts connected to ARPAnet, the first internet, could have thought that what they started could change applications and human life forever. The original hosts on ARPAnet simple shared resources among research facilities and a few University of California schools, so every other connected host could be trusted and help accountable for anything that went wrong [ARPA]. Today, however, is a much different story.

With new opportunities come new challenges. The internet not only makes it easier to communicate with people around the globe for the betterment of humankind, it also makes it easier to engage in activities of questionable ethics. The anonymity of the internet may be of great amusement in a chatroom, but it can be a nightmare when trying to assign accountability when a crime has taken place. Forensic techniques can only take you so far if your "witnesses" all say they didn't see anything; but this situation is all too common when trying to track down instances of fraud on the internet.

As if anonymity were not enough, speed is also a double not-so-hidden danger. Not only can a potential attacker have worldwide connectivity, one can also execute attacks at increasing speeds [0wn]. Broadband connections, from cable to T-1, are being installed in homes and entire apartment complexes making hi-speed internet connections more common.

Malicious Code

Different types of programs have been made to do a variety of different things, such as attack entire networks or hijack personal computers. Even though the pieces of software and the techniques are different, most of them share a common thread – they force their way onto a user's' machine. They can do this in a variety of ways. One way is to send information that is too big for what a program is designed to handle. Since the program must store the data somewhere, the "extra" data that it was sent usually overwrites something already on the user's machine. The "extra" information often provides a means for an attacker to have a "way into" the machine. This type of an attack is called a buffer overflow exploit.

Once in, the software usually "phones home" and downloads the most current version of itself so it can continue carrying out its intended purpose. Once it establishes a firm hold on the machine and has updated itself, if can lie in wait for a future coordinated event involving other compromised computers, or start spreading copies of itself right away.

Not all malicious code forces its way onto a user's machine. In many cases, the user allows it either blindly or knowingly. Some malicious software or software that is designed to present advertisements (Adware) to a user comes bundled with software that the user wants. With the rise of Peer-to-Peer file sharing programs, many developers have chosen to bundle their adware with the desired programs. And since, for the most part, users cannot be bothered with reading how a software package is going to be installed, they either opt for the "express install", which can install just about anything, or blindly click "Yes" to any and all questions asked them, just so they can use the software.

In addition to software packages, users also are bombarded with advertisements while surfing the web. During this time advertisements, which are designed to attract attention by nature, lure users into clicking anywhere within the advertisements window. Once clicked, a variety of software can be downloaded and installed on the users machine when they visit the resulting website.

For Fame and Fortune

People with the technical know-how create tools to do their biding, but sometimes there is no harm intended at all and a curious user tries the tool for fun. On another level, people use and develop hacking tools for bragging rights or to build a reputation in the hacking community. On the highest level, entire criminal organizations develop their own custom tools to deliver attacks, or setup a means to deliver an attack, in

order to extort money from a company who does not want their web services shut down for a period of time. In many cases, paying the crime organization is far more beneficial to the company, since having their web services stop would put them out much more money.

Pharming & Phishing

Gaining information about people on the internet is a key factor in the realm of identity theft and making false accounts for fraudulent purchases. One direct way, that plagues online services such as America Online, is to simply ask the user for usernames and passwords. Even though services such as AOL explicitly tell their users never to give away their account information, this technique of asking masses of users for their personal information, called phishing, has been very successful.

Another technique, called pharming, is one where raw data is "harvested" from many users by making them believe they are submitting information to a legitimate website [IPU]. The data is then sorted and assembled into a "refined product". The raw data can be such things as login names, passwords, addresses, credit and bank card information and possible pin or security codes. The refined product can either be a user's entire identity or a fake identity which will be used purchase goods with someone else's money (a.k.a. "carding") or simple to make quick money by washing it through some other means.

Resource Exploitation

As patches and awareness increases, coders get cleverer and use "free" resources in a way which they were not intended. Many sites used to host pictures, journal entries and file storage are being used to store and distribute malicious code. Also, users' computers themselves are also being used in massive unison to attack a single target. Although a single computer asking a web host a simple message 10 times a second is trivial, 500,000 computers compromised at an earlier date all asking the web host in the same manor can quickly flood the lines leading to the web host with only those messages. And since the host is processing those "attack" messages instead of normal traffic, the non-malicious users cannot be serviced, which is why this kind of attack is called a denial of service attack [AT&T].

Moving at the Speed of Slow

While spyware and adware plague the internet, increasingly better tools are being developed to combat the problem, many of them free. But that does not solve the problem. Since there are legal issues involved with worms that crawl around the

internet clean computers then delete themselves without a trace, users still have to actively download tools to help them take care of their computers. Although companies like Microsoft try and combat this problem by having "Automatic Updates" and release security patches every moth, a lazy user who refuses to take the time to download the newest security update can continue to spread a disease that already has a cure.

It is not always laziness that stops users from upgrading. Sometimes the fear of having a security patch make something else malfunction is not worth the upgrade. For example, a critical patch for Microsoft Internet Explorer could stop the transmission of malicious code, but could disrupt company operations because of how it changed the functionality of the browser [Kotoda].

This slow upgrade process lengthens the life of many malicious software packages artificially. Sometimes software distributors require an update or patch after a period of time or disguise it by coming out with a whole new version of the user interface. This can force true users of the software to finally upgrade, thereby decreasing the possible number of computers that can be infected by a "cured" vulnerability.

Broken Software Practices

Software practices have not kept pace with the new emerging threats that are posed by the Internet. These practices are rooted in the training and biases of the developers, the tools and languages they use as well as the interest of the business that drive the software industry.

Software Engineering Biases

Developers have traditionally been given the wrong incentives for producing secure code. Generally, developer productivity is measured in lines of code, function points or features delivered. Quality is measured on number of bugs (not specifically security bugs). Once software is shipped, little accountability for the mistakes are tied back to the person who created them. Developers are also quick to code and slow to design and even slower to identify security threats. Most software engineering approaches prescribe that testing be performed based on a scope derived from the requirements specification or use case analysis documentation. Test scripts are compiled from test cases that trace back to the each of original requirements. This traditional style of testing demonstrates that an application verifies and validates the specified functionality of the system. By testing only against what is specified in the requirements, any segment of the application outside that specification goes untested. A large portion of security vulnerabilities is found in the extra-specification portion of

the code. A fundamental shift in testing mindset is required—moving from testing required based on solely on requirements and use cases to one that also incorporates threat scenarios and "misuse" cases [Pauli].

Software Tools and Languages as a Source of Security Problems

Applications are written as if all users and the environment were cooperative. Operating systems such as windows were created with security as an afterthought. The internet was developed within DARPA and US military as a system that wasn't about central control, but about being able to survive node loss during nuclear war. C and C++ (as well as other languages) and their runtimes were created to allow the full creativity of users who know what they are doing. Because the C programming language evolved from a typeless language it carries idiosyncrasies that make it more vulnerable to security problems [Ritchie]. Specifically, its syntactic similarities between pointers and array types and how strings/buffers are manipulated has made it more vulnerable to buffer overruns. If arrays' bounds are checked, many hacks could be avoided. [Bates] The conciseness and expressivity of C/C++ often makes it difficult to detect simple mistakes that can be exploited by malicious users. It is hard spot mistakes in code when the differences between valid and invalid statements in these languages are so close together. A misplaced semicolon can transform a loop into a block of statements that is executed only once [Chinchani]. Also, the runtime libraries can be difficult to always use correctly. For example, developers are forced to explicitly allocate and deallocate memory on the heap. Any mistake in how this is performed can introduce can lead to "double free" bugs.

Lack of Training in Secure Development Processes

Most computer science or computer engineering students have not been adequately trained in security practices for software development. Computer science programs in universities require some software engineering coursework. These programs generally emphasis how to achieve quality (of which security is only one aspect) and rarely address security explicitly. Classes may include project management, requirements gathering, design, and testing, but don't address specific issues of software security [JTFCC]. Even if graduating computer science students receive training in software engineering, not all people entering the information technology workforce have degrees in CS. About one-third of people working as programmers have degrees in computer science, and about one-quarter of those in IT jobs employment hold computer and information science degrees. Most candidates hold degrees in a wide range of fields such as engineering and mathematics to psychology and education [USDoC].

Conflicting Interests of Software Marketplace

The current software development marketplace is driven by demands for new features. Security of the software is generally taken for granted by the user and is difficult to impossible to take into consideration when a software package is being considered for purchase. Because of this, most product releases are devised on the next se of features that can be added based on what the market demands. While product development understand that reputation could be lost if a product has low security quality, rarely does the business put security priority over features.

Improved Software Processes

There are software processes that can be used to prevent critical security bugs from ever being introduced into an application. Companies that are willing to make improvements to their processes are already empowered to be able to adopt some of these these techniques. The recommendataions in this section are for that type of fundamental industry process. They are organized according to the phase of the software lifecycle that they are relevant to: the design phase, the implementation phase or the maintenance phase.

Design Processes

Testable Designs

It has been mentioned that a hacker mentality can cause software developers to produce applications with low quality. A paradigm shift by developers is required to alleviate this problem. Traditionally software developers view themselves as having complete freedom in the way that they design and implement software. There is a hubris which leads to a cowboy mentality which leads to shortcuts which leads to expensive defects [Psych]. New design paradigms clearly emphasize that this problem can be solved with designs that are simple and testable. A simple design ought to be favored over a complex one, even though the complex one may be believed to be slightly more efficient. A design ought to take into consideration the ways that the design can be tested [Complete]. An attempt should be made to verify the correctness of the design, and this is only a tractable problem if the design was kept simple.

Security-sensitive components of applications can benefit even more from this principle. A simply designed security component can be verified to be correct, but a complex design could hide vulnerabilities for a long time.

Programming Languages

One tool for writing secure software is a reliable programming language. Most software could be written using any of several modern programming languages. The decision on which language to use might be based on efficiency or developer familiarity. Some languages are markedly different in their reliability for secure applications, however [Howard]. In newer languages like C#, it is easy to avoid both the double malloc and buffer overrun bugs. Mistakes can still be made that will lead to an exploit using a language like C#, but it is materially easier to write safe code. A desirable language trait is that security-sensitive operations are easy to spot and easy to review in software. Another desirable trait is that inexperienced developers will be able to write secure software. A corollary to this is that only experienced developers should know how to expose the features that could lead to security vulnerabilities, and since they are experienced developers they will know how to safeguard themselves. In contrast, many fundamental and commonly-used C functions (such as "malloc") are needed by beginners but are hard to use safely.

Companies that wish to ensure that their software security improves can move towards languages that were designed with security in mind. This move does not prevent security defects, but it can turn large classes of bugs into a rare occurrence. This move requires a large upfront cost for developer education and it also requires the discipline to make a language decision based on the security implications rather than other factors.

Threat Modeling

It is accepted by security experts that software has adversaries and that it has to be designed accordingly. The process of threat modeling assimilates that outlook into the design review process. Designs can be evaluated based on a threat model analysis. Using a threat model analysis, design flaws can be corrected early in the software development process before they ever become security holes [Howard]. Another advantage to threat modeling is that it encourages architects and developers to change designs so that vulnerabilities are removed entirely. Once the software is designed and implemented, it is usually impossible to remove vulnerabilities, and there is only time to mitigate them. An example of mitigating a vulnerability is adding input validation tests to ensure that a function does not operate on unexpected data. An example of removing a vulnerability is to remove the function entirely and alter the data flow so that the work performed by the function is no longer necessary. Sometimes this can be done by reducing feature sets or by discovering more elegant ways to implement the required functionality.

Threat modeling follows some steps for making sure that damage that can be caused by an attacker is mitigated. First the assets or resources that are at risk are identified.

Then the means by which an attacker could compromise those assets are identified. The damage that would be caused by each possible attack is rated according to its severity. Note that vulnerabilities are examined, not exploits. A vulnerability means that there is a possible attack vector, whether there is a known exploit or not. Each vulnerability is treated as if there were a working exploit that will be discovered. Then possible mitigation techniques are identified. All vulnerabilities should be mitigated somehow. The most severe vulnerabilities should all have strong mitigations where multiple successful attacks would be required in order to compromise the assets.

Defense in Depth

The mantra of "defense in depth" is another paradigm shift that can be adopted by companies and developers that would like to reduce security exploits. With this mindset, it is assumed that there could be an as-yet undiscovered security hole in any part of an application. In order to prevent damage, security checks are placed in multiple places. Sensitive security assets should be protected by multiple layers of security [Howard]. Even though each layer individually should be designed to be sufficient to guard against attacks, multiple such layers offer the strongest safeguards. An application that exhibits defense in depth can sometimes protect all its sensitive assets even if attackers are successful at compromising individual components or layers of the system.

Implementation Processes

Static Analysis Tools

Static analysis is the process of verifying that applications are free of defects by examining the source code directly. Static analysis can only be done with sophisticated tools. These tools scan through source code for signatures of known defects. In this sense they are similar to virus scanners which scan files for patterns that identify known viruses. The first step is for security experts to identify common source code idioms that are unsafe. For example, most buffer overrun bugs are caused by a defect where the buffer size is not tracked scrupulously. These types of bugs can be found by scanning source code for buffer operations that do not take the correct size into account. The signature for these types of bugs is programmed into the static analysis tool so that it will always be able to flag instances of the bug. Then the tool must be run regularly and the flagged source code must be fixed.

Static analysis is more reliable than code reviews because the tools can scan millions of lines of source code without getting tired or making a mistake. The tools can only be used to find relatively simple signatures, however. Many common security bugs,

such as those related to buffer overruns, can be prevented like this, but more exotic bugs are beyond the reach of current static analysis tools [Howard].

Code Instrumentation

Another technique that is related to static analysis is code instrumentation. Software can automatically be made more defensible by automatic means. The best example of this is the "/GS" compiler flag [Howard].

The "/GS" flag [GS] is another defense against stack smashing attacks. The compiler automatically generates code that checks against buffer overruns. If a running program is attacked and if the stack is smashed, then the generated code will shut down the process before any other assets are compromised. This type of defense works best in concert with other buffer overrun mitigations.

Security Reviews

The practice of regular security reviews is known to increase the quality of software. Security reviews are expensive, but they are still less expensive than deploying the fix for a security defect. Some companies already institute security reviews as part of their standard development practice. A security review is when an expert (but not the expert that designed or implemented the software) examines an application to find potential security holes. Security reviews focus only on security-sensitive issues and ignore the thousands of other issues that affect software quality. Sometimes the reviewers will be following a checklist of items in order to make sure that all the items on the checklist were remembered by the developer. Sometimes the security review is part of a quality gate that the product is required to pass before it can be shipped to customers [SDL].

Security reviews institutionalize a desire for security that most application development teams already have. The security review gives developers a chance to concentrate only on finding security defects, and so the quality of the security analysis will be improved. The security review also gives developers the goal of writing software well enough that it passes the security review the first time. These factors serve to remind developers of the importance of security, which can help teams make better decisions when tradeoffs arise between developing more features or developing more secure features.

Maintenance Processes

Root Cause Analysis

Root Cause Analysis (RCA) is the process of searching for underlying reasons for failures. A root cause analysis is more in-depth and more expensive than a traditional analysis.

A traditional analysis usually includes the minimum amount of work that needs to be done in order to return the software to a trusted working condition. The first step is that unexpected behavior in the application gets reported. Customers and maintenance engineers try to reproduce the problem and to record the steps that are necessary to trigger the bug. Then regression tests can be written that isolate the problem by executing a small number of steps that will cause the bug to recur. A failure in an application typically can be traced back to a single code defect. The code defect is fixed by developers. Then the fix is packaged and shipped to the customers. The customers verify that the modified application works correctly in their environment and they deploy the new version.

A root cause analysis adds several steps to the traditional process. Before performing a root cause analysis, all the traditional work of isolating and fixing the bug must occur. In addition, the analysis includes steps to prevent bugs of a similar nature from happening again. A typical technique is known as the "5 Whys" [XP]. Using this technique, an investigator asks why the bug occurred. Then the investigator takes the answer and asks why that was the case, repeated about five times. Some bugs require more or fewer questions to arrive at a useful answer. The goal is to use the information gleaned from one failure in order to prevent future failures. Using this technique, each discovered failure can lead to the software becoming more reliable rather than less reliable.

Sharing Knowledge

The current state of the art in software development is that developers must learn little by little over the course of an entire career about good practices. There is not a central repository of knowledge for good software practices and procedures. In fact, there is not much agreement on which practices are desirable nor on the standards that determine what makes a practice good. In this sense, software development is more of a craft than an engineering discipline. Because of its craft-like nature, software engineers all over the world have to continually relearn the same lessons from the same mistakes, even though other engineers had already discovered that knowledge.

One of the distinguishing features of other engineering disciplines, such as civil engineering, is that there are codes that prescribe and govern correct practices. The knowledge necessary for creating these codes still needs to be accumulated in the

computer science field. To this end, cooperation is needed between corporations and universities to share knowledge and best practices. These organizations ought to continue to work together with the goal of removing the craft and superstition from software development.

The Role of Public Policy Governing the Changes

In addition to the industry processes that have been described, there are some improvements that are dependent on public policy changes. Government and industry are already cooperating on some fronts to improve software. A few of these ongoing initiatives will be examined here.

The Beginning of Computer Security

Public policy for the protection and security of computer has dated as far back as the year 1973. At this time, many were evaluating the impact of storing personal medical records on computers. Many wanted to gain from the benefits of computerization, but at the same time they were concerned with the safety and reliability of storing personal information on computers.

Not much has changed since then, but now security has become more complicated and at the same time hackers and cyber terrorists have become more sophisticated. Computer technology has become so advanced that consumers have also put more trust in computers by not only having medical data digitally archived, but engaging in such activities as online banking and purchasing products online with credit cards. These activities that deal with private information has become a target to cyber terrorists. The sophistication of their attacks has occurred not only because of knowledge of vulnerabilities, but because developers fail to make security a top priority when it comes to the development of software. Policies must be enforced to prevent the furthering of software attacks. Some policies that the past and the present still abide by are the Code of Fair Information Practices, which consists of five clauses: openness, disclosure, secondary use, correction, and security.

Licensing Software Engineers

Many argue that the security failure in software is caused by inadequately trained developers. Although they are able to produce software that can provide some basic functions, many fail to take notice of complications that emerge when coexisting with other programs, the limited information from testing, and other specifications that require high standards to create a stable and secure software. Their concerns are for the bare minimums, not for overall stability. Requiring licenses from software

developers could prove to be a benefit in that the minimum requirement is a high standard of knowledge in not just creating a basic function software, but one that reliably secure.

In dealing with licenses, many other professional fields follow this process to maintain an educated and qualified work force that leaves little doubt as to their capability. Such fields that are comparable are lawyers and other engineering divisions because of the extensive steps one must take to be licensed. In order to be a lawyer or engineer of other divisions, one must complete education at a legitimate and accredited institution, pass a number of challenging examinations, and also have some form of experience by means of internships or working in a mentor-disciple environment. Without licenses, software developers are basically similar to journalists where no education or high standards are needed to get ones work published. As of today's standards, the every day blogger to a lonely pamphleteer is considered a journalist, whether or not it is of good quality or not. Software developers are headed in this direction where the average person could become one if he or she chooses without receiving proper training. This results to an inconstancy in quality of work put out.

By enforcing licenses, the internet community and users will be able to considerably distinguish and identify who is more qualified to produce software. Although this method will not necessarily identify the qualified from the unqualified, it gives some level of standard on how to select a higher quality product because a high standard is already set, which can readily lead to less cyber attacks. Licenses could also expire, to ensure that the developers are under constant review. This would establish a standard that would keep all developers up to date with any complications, because in order to renew one's license, one must be aware of current issues. This would ensure that no developer is out of date with current systems.

Expanding Curriculum in Institutions

As stated earlier, there is growing number of computer science or computer engineering students who lack the knowledge of developing security measures for software. Perhaps the universities and colleges of the United States should take a lesson from other universities across the globe such as the Katholieke Universiteit of Leuven in Belgium. This university has developed a course that aims at educating software developers about secure software.

In this course, a wide variety of students are in attendance. Students range from those trying to earn their masters to their PhD, to IT professionals and practitioners. This student base variety allows room for learning as well because different levels of

education on software are aware of different issues. This gathering of information of what is known at some levels allows for others to learn and be informed as a whole.

This course that is taught in Belgium varies not only in its students, but topics that are taught as well. This course educates students by using a number of case studies to learn from, understanding code and cryptography, and ideas on how to develop technologies that have attained a high degree of quality.

By adopting a similar curriculum in the United States and internationally, the foundation of the software developing community could possibly improve and become a lot stronger. Having the developers more aware of issues such as security and by having security issues at the forefront could create some fierce competition that would in turn create better and more stable products.

Private Sectors Taking Action

While cyber security is not exactly at the forefront of governmental affairs, many groups are trying to change that by focusing on cyber security. One group is the Cyber Security Industry Alliance (CSIA) who is taking a step forward by making cyber security their primary objection and focus. The CSIA, which was formed in February of 2004, "is an advocacy group dedicated to the improvement of cyber security through public policy, corporate outreach, education, awareness, and technology-focused initiatives" (Cyber Security Industry Alliance). The CSIA mission consists of four goals: 1) public policy, 2) education, 3) awareness, and 4) standards. With public policy, the CSIA hopes to influence the decisions in law making in both the state and federal level. Their goal for education is to work with leading institutions and create courses that enhance the skill of developers. For awareness, the CSIA hopes to inform not only corporations, but small businesses and the consumers about issues that emerge dealing with security. And last, but not least, with standards they hope to work in conjunction with other affiliations to advance computer standards.

Although it appears that the CSIA is headed in the right direction, it comes at a high price. In order to be a member of CSIA, one must pay annual dues ranging from $12,000 to $150,000. Membership is not solely based on the desire to advance in computer security, but also on financial benefits one can offer. One could argue that there is a need for such dues because all activity requires some finances, but this is a disincentive to smaller business who would otherwise agree to be member if the membership did not have such a high price. CSIA does offer some benefits such as uniting many technology companies, but one must pay to be part of a united force.

Government and Self Regulation: EU Safe Harbor

In October of 1998, the European Commission's Directive on Data Protection began its application of banning the "transfer of personal data to non-European Union nations that do not meet the European 'adequacy' standard for privacy protection" (US Department of Commerce). Unfortunately the United States did not meet up to such high scrutiny, and many transactions were lost. The United States shared the same vision of creating some standard of protection for data, but most of the standard heavily relied on "self-regulation". The European standard was heavily relied on government created programs and agencies that were under high scrutiny. These different standards lead to conflicts and obstacles of business between Europe and the United States.

The EU Safe Harbor aimed at solving that problem. In the year 2000, the Safe Harbor went into effect. This implementation ensured US and European companies that business transactions between the two countries would have "adequate privacy protection". Members from both sides gain the benefits of knowing that those whom they are doing business with are must meet a high level of scrutiny. This ensures that the majority of transactions will be valid and with out complications.

This initiative is more accessible in that it that it requires no annual fee and to be a member requires only the promise to abide by the Safe Harbor principles. The absence of an annual fee allows more companies to join, proving it is not an exclusive club, but rather an initiative that aims to create an environment where companies can make transactions without worry.

The EU Safe Harbor is headed in the right direction when applied to secure software. The Safe Harbor aims at creating some standard of ethics on an international level. The principles of the Safe Harbor are notice, choice, onward transfer, access, security, data integrity, and enforcement. This standard of ethics on an international level is something that the security of software needs. The Safe Harbor not only requires overall abidance of the principles, but focuses on having safe access, that is secured. Failure to apply could lead to membership withdrawal and no access to benefits stated earlier. The principles that require an "adequate" level of security will ensure companies are constantly updating security measures therefore decreasing the risk of attacks or intrusions on personal data. This requires that companies keep themselves on check, but at the same time the government does not force them to abide; it is purely the companies choice if they want to abide. The burden is placed on the consumer of the software and to make sure they are always up to date. The Safe Harbor allows for self and government regulation to coexist.

Government Action: California SB 1386

On July 1, 2003, companies, businesses, organizations, and anyone who does business with a resident of California, is now bound to the law SB 1386, a law that requires them to notify a California resident if there has "any breach of security of the data". This personal information includes: driver's license numbers, social security numbers, account numbers, bank and credit card numbers, or any confidential information about a California resident.

This measure enforces security at a higher level. Those who fail to comply by not informing individuals that their confidential information may have been compromised can be held liable for "civil damages or face class actions". This burden of liability on companies and organizations forces them to maintain a secure system that enables them to be less prone to attacks and security threats. Forcing security to be a priority for companies and organization obligates software developers to focus on security in their software. This domino effect leads to the awareness that security should be a top priority. Having the government enforce security measures is one of the most effective ways to implement policy. Not only will people be informed that security is an issue, but many will be obligated to abide.

California is headed in the right direction with SB 1386, but residents of other states are unfortunately not protected by this law. Perhaps the Federal Government should, enforce a measure similar to SB 1386 on a federal level so that as a nation, many will be obligated to maintain a secure system. This could also have an international effect because there are many business transactions that occur over seas. SB 1386 forces any one who makes a transaction with a California resident to maintain a secure system, therefore any one who does business must keep security a priority.

Conclusion

Strong leadership is needed to make the required improvements in development processes, programming languages, tools and education. Leadership is within the industry can be motivated by market pressures as seen by Microsoft's recent emphasis on security. Where that leadership is lacking, government regulation may be needed to bring about the required changes. While most agree that software cannot be made perfect, improvements in the way software is developed can make it more secure and safeguard the privacy of its data.

References

[0wn] Stanifor, Stuart; Paxson, Vern; Weaver, Nicholas "How to 0wn the Internet in Your Spare Time" http://www.icir.org/vern/papers/cdc-usenix-sec02/

[ARPA] "ARPAnet – The First Internet", http://inventors.about.com/library/weekly/aa091598.htm?pid=2821&cob=home

[AT&T] Ioannidis, John, "Distributed Denial of Service" – AT&T Labs Research, http://www.tla.org/talks/ddos-ntua.pdf

[Bates] Bates, Rodney "Buffer Overrun Madness", Open Source Vol 2, No 3, May 2004.

[Chinchani] Chinchani, R.; Iyer, A; Jayaraman, B; Upadhyaya S, "Insecure Programming: How Culpable is a Language's Syntax?" Proceedings of the 2003 IEEE Workshop on Information Assurance, June 2003

[Complete] McConnell. "Code Complete." Microsoft Press, Redmond 1993.

[CSIA] Cyber Security Industry Alliance "Goals" 2005 https://www.csialliance.org/ourwork/goals/

[CSIA] Cyber Security Industry Alliance "Membership Benefits and Priveleges" 2005 https://www.csialliance.org/membership/

[GS] Bray, Brandon. "Compiler Security Checks in Depth." http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vctchcompilersecuritychecksindepth.asp

[Howard] Howard, Michael; LeBlanc, David. "Writing Secure Code," Second Edition. Microsoft Press, Redmond, 2003.

[IPU] "Pharming" - http://pharming.area51.ipupdater.com/

[JTFCC] LeBlanc, Rich; Sobel, Ann, Joint Task Force on Computing Curricula, "Software Engineering 2004: Curriculum Guidelines for Undergraduate Programs in Software Engineering" August 23, 2004.

[Knight] Knight, John C "Should Software Engineers Be Licensed?" Department of Computer Science, University of Virginia http://www.cs.virginia.edu/~jck/publications/licensing.position.for.web.site.pdf

[Pauli] Pauli, Joshua J.; Xu,Dianxiang "Misuse Case-Based Design and Analysis of Secure Software Architecture" Proceedings of the International Conference on Information Technology: Coding and Computing, 2005

[Piessens] Piessens, F.; Jacobs, B.; Joosen, W. "Software Security: Experiments on the .NET Common Language Run-Time and the Shared Source Common Language Infrastructure" October 2003

[PRC] Privacy Rights Clearinghouse "A Review of the Fair Information Principles: The Foundation of Privacy Public Policy" February 2004 http://www.privacyrights.org/ar/fairinfo.htm

[Psych] Weinberg, Gerald M. "The Psychology of Computer Programming," Silver Anniversary Edition. Dorset House Publishing, New York, 1998.

[Ritchie] Ritchie, Dennis M "The Development of the C Language" presented at Second History of Programming Languages conference, Cambridge, Mass., April, 1993. http://cm.bell-labs.com/cm/cs/who/dmr/chist.html.

[SB1386] Peace, Steve; Simitian, Joe "SB 1386 Senate Bill – Chaptered" February 2002 http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html

[SDL] Howard, Michael. "A Look Inside the Security Development Lifecycle at Microsoft." MSDN Magazine, November 2005.

[ThreatFocus] "Threat Focus California SB 1386 Compliance" http://www.threatfocus.com/sb1386.php

[USDoC] Export Portal, US Department of Commerce "Safe Harbor Overview" http://www.export.gov/safeharbor/sh_overview.html

[USDoC] Office of Technology Policy, US Department of Commerce "Update: America's New Deficit" January 1998.

[XP] Beck, Kent. "Extreme Programming Explained: Embrace Change," Second Edition. Addison-Wesley Professional, 2004.

[ZDNet] "Alarm over 'pharming' attacks" http://reviews.zdnet.co.uk/software/internet/0,39024165,39188617,00.htm

[Kotoda] Munir Kotoda - Microsoft IE Patch Leaves Users Locked Out - February 2004 - http://news.zdnet.co.uk/internet/security/0,39020375,39145482,00.htm