Volume 9, Issue 1
April 2016
ISSN: 1946-1836

Journal of
Information Systems Applied Research

In this issue:

The **Journal of Information Systems Applied Research** (JISAR) is a double-blind peer-reviewed academic journal published by **ISCAP,** Information Systems and Computing Academic Professionals. Publishing frequency is currently quarterly. The first date of publication was December 1, 2008.

JISAR is published online (http://jisar.org) in connection with CONISAR, the Conference on Information Systems Applied Research, which is also double-blind peer reviewed. Our sister publication, the Proceedings of CONISAR, features all papers, panels, workshops, and presentations from the conference. (http://conisar.org)

The journal acceptance review process involves a minimum of three double-blind peer reviews, where both the reviewer is not aware of the identities of the authors and the authors are not aware of the identities of the reviewers. The initial reviews happen before the conference. At that point papers are divided into award papers (top 15%), other journal papers (top 30%), unsettled papers, and non-journal papers. The unsettled papers are subjected to a second round of blind peer review to establish whether they will be accepted to the journal or not. Those papers that are deemed of sufficient quality are accepted for publication in the JISAR journal. Currently the target acceptance rate for the journal is about 40%.

Questions should be addressed to the editor at editor@jisar.org or the publisher at publisher@jisar.org. Special thanks to members of AITP-EDSIG who perform the editorial and review processes for JISAR.

2016 AITP Education Special Interest Group (EDSIG) Board of Directors

Journal of
Information Systems Applied Research

Editors

JISAR Editorial Board

Moving Beyond Coding:
Why Secure Coding Should be Implemented

Mark Grover
mjgrover@us.ibm.com
Technical Enablement Specialist, Watson University
IBM Watson Group
Durham, NC

Jeff Cummings
cummingsj@uncw.edu

Thomas Janicki
janickit@uncw.edu

Information Systems and Operations Management Dept.
University of North Carolina Wilmington
Wilmington, NC 28403

Abstract

Consistently, malicious attacks through unpatched software continues to be one of the leading causes of security breaches year after year. Most attention has been placed on continuous patching to eliminate any security holes in existing software. However, as more devices continue to be connected (i.e., Internet of Things) and entire industries move to a connected environment (e.g. healthcare), closer attention needs to be placed on the development process, specifically implementing secure software development guidelines. The purpose of this research is to examine the current security issues related to inadequate focus on secure coding and to provide an overview including suggestions on how to improve coding by focusing on security during development. In the following paper, we discuss the need for secure coding by first evaluating current data breaches caused by software flaws followed by a history of secure coding. This is followed by a discussion options available to developers for implementing secure coding. We finish by providing general recommendations for incorporating secure coding into current practices that could be adapted for both an organizational environment and higher education.

Keywords: Software Development, Secure Coding, Hacking, Certified Ethical Hacker

1. INTRODUCTION

Companies, governments, and private individuals are more and more vulnerable to the loss of confidential information. This is exasperated as companies are conducting more of their B2B, B2C and internal applications via internet or cloud applications. Loss is not limited to organizations as the exposure to loss of personal information for private citizens continues to grow exponentially as the number of smart devices grows (National Vulnerability Database, 2015). While the reasons often mentioned for these losses include poor security policies, network intrusions, hardware swapping, vendor / supplier lack of security policies (Patrizio, 2014), poor software programming is often cited as how information us ultimately stolen (Verizon Solutions, 2014).

Much has been written to address the need for CIO and CTO's to address the protection of data (Pettigrew, et al. 2010; Richardson, 2008; Zafar, et al. 2011). This paper will review one potential area that all software developers could undertake to improve the quality of their coding and make their applications more secure. Jones and Rastogi (2004) argued as far back as 2004 for the need for building security into the software development life cycle.

To understand the need for secure coding, the paper will initially review significant breaches of data as related to software coding issues. This will be followed by a discussion of the term and definition of 'secure coding' including concepts of what makes coding more secure. Next the concept of certification and training in the area of Certified Secure Programmer will be introduced. Finally, recommendations will be made to software developers and their supervisors will be made.

2. DATA BREACHES

Announcements from firms experiencing a data breach have become a daily occurrence. This includes many of the top breaches recently reported including JPMorgan Chase, Target, Home Depot, SAP and numerous companies impacted by the Heartbleed bug (see below for additional details of each breach). Many of these breaches can be traced back to a vulnerability discovered in a piece of software implemented at the breached organization. In the following section, we will highlight a few of these recent breaches expanding on how most were caused by vulnerabilities in software that could have been minimized by secure coding.

JPMorgan Chase

An attack occurring at JPMorgan Chase late last year resulted in the compromise of 76 million household accounts and 7 million small business accounts. The cause of the breach has been traced to hackers obtaining a list of applications being run at the organization which was then crosschecked with known vulnerabilities (Silver-Greenberg, Goldstein & Perlroth, 2014). While there was no evidence that financial data was compromised, JPMorgan Chase did alert customers that names, addresses, phone numbers and emails were likely stolen.

Target & Home Depot

Both attacks on Target and Home Depot can originally be tracked back to access through third-party vendor credentials. The malware was then loaded on the companies POS terminals which enabled hackers to steal the customer's credit card information totaling 80 million and 20 million, respectively. Recent details suggest this malware was exploiting a software vulnerability in the operating system installed in the POS system (Patrizio, 2014).

SAP

Recently, researchers analyzed hundreds of companies who had SAP implementations and found that over 95% of SAP deployments are vulnerable to cyber security attacks. Many of these companies also had patching windows of over 18 months. This is critical as in 2014 alone, SAP average more than 30 patches a month (~391 security patches issued) (Curtis, 2015).

Heartbleed Bug

Finally, in 2014, the Heartbleed bug which affected OpenSSL sent companies scrambling to patch the security hole in the software with some estimating close to 66% of active sites on the Internet being impacted. These included companies such as Google, Facebook, YouTube and Amazon (Schneier, 2014).

The breaches just described include only a few examples in which security flaws found within software can lead to breaches. While these have been highly publicized breaches, they are not the only cases of breaches as the number of vulnerabilities found in software continues to rise. The national vulnerability database reported that, in 2014, the category of software flaws included over 7,900 identified vulnerabilities and at the end May 2015, over 2,500 software flaws have been identified causing numerous vulnerabilities in a variety of common software used at organizations (figures obtained from the national vulnerability database, https://nvd.nist.gov/).

Many researchers expect vulnerabilities to continue to increase exponentially as we connect more and more devices. This includes anything from medical devices (Forrester, 2015) to home devices such as watches, activity tracker (e.g. Fitbit) and other devices falling under "Internet of Things" (Hesseldahl, 2015). As we expand into these new areas yet to extensively explored, it becomes imperative to increase the focus on secure coding to stop vulnerabilities at the source. The subsequent section discusses how secure coding has been approached in the past and what is needed into today's environment.

3. SECURE CODING AND NEED

A review of academic journals resulted in few articles in the area of building more secure applications. To find a clear definition of secure coding, one must examine practitioner material such as the application developers' guides and other material distributed by the major application platforms.

Apple's Developer Library defines secure coding as "the practice of writing programs that are resistant to attack by malicious or mischievous people or programs." (Apple, 2014) Jim Canup, Enterprise Security Consultant for HP's Fortify Software Company, defines secure coding as "a process used to decrease risk and increase the overall quality of code as it pertains to security." (Canup, 2012) Microsoft's definition is quite simple, "write code that can withstand attack and use security features properly." (Microsoft, 2014)

Combining their definitions one could define secure coding as:

> "The practice of writing code that is resistant to attacks."

If the leading companies understand the need for increased concern from security in coding, why is it not always done? According to Kenneth Van Wyk (2003), there are three factors that work against secure coding: "*Technical factors* (underlying complexity of the task), *Psychological factors* ('mental models'), and *Real-world factors* (Economic or other social factors)." The real challenge is that coders typically work for profit and have limited time and resources available to complete a given task. Best intentions, and practices, are often challenged when faced with deadlines.

History of Secure Coding

Software development has been around for over 60 years, and this begs the question, how long has secure coding been practiced? It appears the appreciation for the need for secure coding only came with the explosion of the internet and the .dot com era. Prior to the internet most coding efforts were for individual companies and access by malicious outsiders to company software was very limited.

For Microsoft, Bill Gates elevated the need for increased secured code in early 2002. On January 15, 2002, Bill Gates sent out an email to all full-time employees at Microsoft detailing the company's highest priority for the year. In the email Bill Gates states, "Trustworthy Computing is the highest priority for all the work we are doing. We must lead the industry to a whole new level of Trustworthiness in computing." (Gates, 2002) He goes on to define "Trustworthy Computing as computing that is available, reliable and secure as electricity, water services and telephony." Gates goes on continues to list the three key aspects as being Availability, Security, and Privacy.

As a result of this key priority, in January 2002, Microsoft formed the Trustworthy Computing team which was responsible for the development of the Microsoft Security Development Lifecycle (SDL) (Microsoft, 2014). The Microsoft SDL became a mandatory policy in 2004, and now is an integral part of software development at Microsoft. Microsoft's then director of Trustworthy Computing, Tim Rains said "Organizations today simply cannot afford to conduct business online without prioritizing security." (Rashid, 2013). Microsoft has been pushing its Security Development Lifecycle since its inception, and makes the tools and resources freely available. A Trustworthy Computing Blog entry titled "SDL at 10: Driving Business Vale", dated March 6, 2014, states that to date Microsoft's SDL tools have been downloaded over 1 million times. (Hall, 2014) Microsoft has even created a version for Agile. For a timeline of Microsoft SDL evolution, and a graphical representation of Microsoft's SDL, see Appendix A.

At the same time other software firms were venturing into more secure coding. As far back as 2002, Symantec detailed five problems (and solutions) that "make up 90% of all security vulnerabilities" (Wong, 2002). They are:

1) Buffer Overview - avoid by checking the length type of input data
2) Format String vulnerabilities – avoid by proper input validation and exception checking
3) Authentication – use 8+ character passwords including alphanumeric and special characters
4) Authorization – ensure it is properly performed, avoid falsified data, and check for canonicalization errors (common character set).
5) Cryptography – avoid custom-built cryptographic algorithms

Wong (2002) continues by suggesting some best practices for secure coding include distrusting user input, always using input validation, and using source code analysis to enhance security. The article closes by saying "it would be negligent to not build hacker resistant code." Burnett and

Foster (2004) also addressed practices to incorporate into all applications by examining a particular vulnerability which is the use of client side validation on entry of data. The challenge of client-side validation is that it can be 'easily disabled or custom tools can be used to bypass validation.' Fortunately, more modern software developer tools have assisted developers in avoiding this potential vulnerability.

In an April 2002 edition of eWeek, Dennis Fisher wrote that "CIOs, growing impatient with security vulnerabilities, are fighting back with language in contracts that holds software companies liable for breaches and attacks that exploit their products" (Fisher, 2002). The reasoning was that placing a monetary penalty for poor coding would effectively force companies to be more careful when coding. Not surprisingly, this article is just one of many placing blame on the people coding programs for their vulnerabilities.

One of the early challenges in the first decade of 2000 was the lack of clearly written standards. As already mentioned, Microsoft was developing their own standard but that would not be publically available until 2004. At that time the solution to securing code was the use of code reviews. The problem with code reviews was that it "is a process without a specific deliverable to a customer, and it often becomes a collaborative effort – without a leader, or an owner" (Hentzen, 2002). Code review was designed to find bugs, not to find security flaws. There was a lot of thought being given to the need to securely code, but consistent, tangible ways were not yet clearly established.

In 2003 Microsoft Press published a book titled "Writing Secure Code: Practical Strategies and Proven Techniques for Building Secure Applications in a Networked World." (Howard and LeBlanc, 2003) Recommendations include to think like an attacker, key considerations are:

1) Software must be written to defend all points as an attacker will choice the weakest point for intrusion.
2) Software coding must defend against known attacks but also consider other entry points by intruders.
3) Software developers 'play by the rules' while hackers have no rules.

It concludes with the challenge that any secure coder should consider: "The Internet is an incredibly complex and hostile environment, and your applications must survive there" (Howard and LeBlanc, 2003).

Fast forward eleven years from Bill Gates' original email, and secure coding is still being discussed. In 2013 a Network World article details that secure coding is still a challenge: "Coding practices could use greater attention to security, according to a survey commissioned by Microsoft last fall. Of 2,726 respondents made up of IT pros and application developers, 37% say their organizations build their products with security in mind. Of the 492 developers in the poll 61% say they don't take advantage of risk mitigation technologies that already exist such as address space layout randomization (ASLR), Structured Exception Handler Overwrite Protection (SEHOP) and data execution prevention (DEP)" (Greene, 2013). Among the reasons cited for not using enhanced techniques, convincing management to spend money to implement risk mitigation technologies was given.

As recent as 2013, a survey detailed the emphasis on secure coding by developers globally. It reported a peak of 79% for India develop with secure coding in mind, down to 61% in the United States and lows of 47% in China an only 33% in Japan. The survey also reported that 76 percent of U.S. developers use no secure application program process. The primary reasons for the lack of a secure program process were "cost at 21%, lack of support and training at 26%, and lack of discussion of the topic at 46%" (Ward, 2013).

How to verify if secure coding was used

According to the National Security Telecommunications and Information Systems Security Policy (NSTISSP) #11, the United States government requires that software products used for national security applications be subjected to formal evaluation prior to their use (NIAP, 2014). This is important during the evaluation of commercial, off-the-shelf application and government off-the-shelf products. These products are typically advertised as being secure, but without third party evaluation, such claims cannot be validated. NSTISSP attempts to ensure a given product meets the Common Criteria Evaluation and Validation Scheme (CCEVS) Program as well as the Cryptographic Module Validation Program (CMVP, 2005).

The CCEVS was created 1985 (and most recently updated in 2012) to create a common criterion for evaluating a given product. The Common Criteria is composed of three parts:

- Introduction and General Model
- Security Functional Requirements,
- Security Assurance Requirements.

A key benefit of the standards is that by implementing a common criterion, software products can be evaluated with the same standard. Countries participating in the Common Criteria Scheme are Australia, New Zealand, Canada, France, Germany, Japan, Netherlands, Spain, UK, and the US. The purpose of using a common criterion is that it allows software developed in any one of these participating countries be evaluated by these standards, and be recognized and accepted by other member countries.

According to the National Institute of Standards and Technology, CMVP focuses on validation of cryptographic modules and cryptographic algorithm implementations (NIST). This ensures that the implementation of cryptographic functions adhere to stringent security standards. The reason why this is so important is to ensure no flaws exist in the implementation of a cryptographic method.

3. SECURE CODING CERTIFICATION

As a response for the need for training and certification the industry has developed certification based on specific languages and/or platforms. Examples of these newer certifications are:

- The EC-Council offers the Certified Secure Programmer in .NET, also known as ECSP. This certification "is intended for programmers who are responsible for designing and building secure Windows/Web based applications with .NET Framework." (EC-Council, 2014)
- Global Information Assurance Certification (GIAC) offers three programming related certifications. Their offerings include certification in Java, .NET, and Web Applications.
- The GIAC Secure Software Programmer-.NET (GSSP-.NET) and GIAC Secure Software Programmer-Java (GSSP-Java) certifications require a candidate "demonstrate mastery of the security knowledge and skills needed to deal with common programming errors that lead to most security problems" (GIAC, 2014).
- The GIAC Certified Web Application Defender (GWEB) "allows candidates to demonstrate mastery of the security knowledge and skills needed to deal with common web application errors that lead to most security problems." This certification stresses that "successful candidates have hands-on experience using current tools to detect and prevent Input Validation flaws, Cross-site scripting (XSS), and SQL Injection as well as an in-depth understanding of authentication, access control, and session management, their weaknesses, and how they are best defended" (GIAC, 2014).

International Information System Security Certification Consortium, also known as (ISC)^2, offers a Certified Secure Software Lifecycle Professional (CSSLP) certification. This certification was designed to validate Software Development Lifecycle security competencies. The CSSLP is targeted at people involved in the Software Development Lifecycle with at least 4 years of proven work experience. The certification shows proficiency in "developing an application security program in your organization, reducing production costs and application vulnerabilities, enhancing the credibility of your organization and its development team, and reducing loss of revenue and reputation due to a breach resulting from insecure software." ((ISC)2)

To gauge if industry has embraced certification in hiring a search of information technology job postings was completed in 2014. On the day of the search there were 80,695 tech jobs listed on DICE.com. Searching by each of the previously mentioned certifications resulted in the following responses:

A search for ECSP (EC Council Certified Secure Programmer) on DICE.com resulted in 1 job listed. Interesting the posting was to teach the concept. A search on Monster.Com on the same day resulted in zero job listings. An investigation of LinkedIn detailed 82 people with the ECSP certification.

Likewise, a search for GSSP (Global Secure Software Programmer) on DICE.com resulted in eight unique postings. A GWEB (Certified Web Application Developer) resulted in four postings while the top certification in the search was CSSLP (Certified Secure Software Lifecycle Professional) yielded 13 unique postings.

While there are few postings listing certifications, it is hopeful to see that perhaps some companies are thinking of security more in the development life cycle process. However, this also shows a lack of concern by organizations when it comes to hiring developers with secure coding experience.

4. RECOMMENDATIONS

Following a review of current industry publications and white papers, a checklist is included below of the items to consider when writing software that will help reduce security concerns. These will help both those in organizations and those instructing on coding (e.g., higher education) a foundation to begin incorporating secure coding in their development process. They are a consolidation of recommendations from: George (2013), IEEE (2014), Mano (2015) and OWASP (2010)

1. Explicitly validate all user input
2. Authenticate all users using a mechanism that cannot be bypassed, the default option should be to deny access
3. Earn or Give, but never assume Trust with suppliers or customers (offloading security functions to a client is a lot less trustworthy)
4. Understand how integrating external components changes your attack mechanisms
5. Use caution with dynamic SQL Queries
6. Pay heed to complier 'warnings'
7. Verify database permissions, especially those with write permissions
8. Identify sensitive data and how it should be handled, sanitize data sent to other systems
9. Design with the ability to isolate or toggle functionality.
10. Verify access to known and tested URLs
11. Send garbage to your application as a test
12. Use Cryptography correctly

5. SUMMARY

The research demonstrates not only the need for developers to begin considering secure coding, but also the need for IT management to encourage and implement secure coding principles in their development life cycle.

In reaction to past incidents, software development companies are beginning to recognize that there is a need for secure coding practices, but the adoption rate is still woefully low. For those choosing a career in coding, a certification seems to be a good investment. According to a Dice.com search of "Java programmer", the salary ranges from $50,000 - $120,000 per year. On average, the same job with the addition of a secure coding certification such as GSSP certification will earn more, approximately $85,000 - $130,000 per year.

Writing secure code may take a little more time, but the long term benefits outweigh the initial time investment. The challenge is that new developers are taught how to code for time, and security is often viewed as something done later. Adopting Secure Software Development Lifecycle practices is not just best practice, but it also makes code that is resistant to attacks. Certification may be viewed as a measureable way to verify a programmer's knowledge of secure coding practices. Secure coding is a choice between doing something poorly or doing it the proper, secure way. It is either you pay now (better development) or your pay (a lot more) later.

Additionally, by providing some high level recommendations/checklist, we hope this paper will encourage companies and instructors of software developers to begin incorporating security into software design.

6. REFERNCES

Apple, Inc. *Mac Developer Library*. 11 Feb. 2014. Retrieved 8 Mar 2014. <https://developer.apple.com/library/mac/documentation/security/conceptual/SecureCodingGuide/Introduction.html>.

Burnett, M., Foster, J. (2004). James Foster. *Hacking The Code: ASP.NET Web Application Security*. Rockland: Syngress Publishing, 2004. eBook Collection (EBSCOhost).

Curtis, J. (2015). "95% of SAP deployments 'vulnerable to cyber attacks'." *ITPro. Retrieved 8* May 2015. *http://www.itpro.co.uk/hacking/24577/95-of-sap-deployments-vulnerable-to-cyber-attacks*

Canup, Jim (2012). "Secure Coding: Best Practices." *Info Systems Audit and Control Association: 2012 North America Computer Audit, Control and Security Conference Resources.*

EC-Council Inc. (2014) *EC-Council Certified Secure Programmer (ECSP) .NET*. Retreived 8 Mar. 2014. http://www.eccouncil.org/Certification/ec-council-certified-secure-programmer-dotnet

Fisher, Dennis (2002). "Contracts getting tough on security (Cover Story)." *eWeek: The Enterprise Newsweekly* 19 Apr. 2002. Academic Search Complete. Retrieved 6 Mar. 2014.

Forrester, A. (2015). "IEEE Issues Security Guidelines for Medical Device Software Development." *ExecutiveBiz.* Retrieved 19 May 2015. *http://blog.executivebiz.com/2015/05/ieee-issues-security-guidelines-for-medical-device-software-development/*

Gates, Bill (2002). *Bill Gates: Trustworthy Computing*. 17 Jan. 2002. Retrieved 8 Mar. 2014. <http://www.wired.com/techbiz/media/news/2002/01/49826>.

George, Randy (2013). The Five Most Common Security Pittfalls in Software Development. Dark Reading, Retreived 3 June 2015. http://www.darkreading.com/the-five-most-common-security-pitfalls-in-software-development/d/d-id/1140103?

GIAC, Inc. (2014). *GIAC Secure Software Programmer-Java (GSSP-JAVA)*. Retrieved 8 Mar. 2014. http://www.giac.org/certification/secure-software-programmer-java-gssp-java

Greene, Tim (2013) . *Microsoft commits to secure coding standard*. Retrieved 8 Mar. 2014. <http://www.networkworld.com/news/2013/051413-microsoft-secure-coding-269733.html>.

Hall, Adrienne (2014). *SDL at 10: Driving Business Value*. Retrieved 8 Mar. 2014. <http://blogs.technet.com/b/trustworthycomputing/archive/2014/03/06/sdl-at-10-driving-business-value.aspx>.

Hentzen, Shil (2002). *The Software Developer's Guide*. Whitefish Bay: Hentzenwrke Publications, 2002. eBook Collection (EBSCOhost).

Hesseldahl, A. "A Hacker's Eye View of the Internet of Things." *Re/code. 7* Apr. 2015. *http://recode.net/2015/04/07/a-hackers-eye-view-of-the-internet-of-things/*

Howard, M., LeBlanc, D. (2003). *Writing Secure Code: Practical Strategies And Proven Techniques For Building Secure Applications In A Networked World*. Redmond: Microsoft Press, 2003. eBook Collection (EBSCOhost). 6 Mar. 2014.

IEEE (2014). Avoiding the Top 10 Software Security Design Flaws. IEEE Center for Secure Design. Retreived 3 June 2015 http://cybersecurity.ieee.org/images/files/images/pdf/FOR_DISTRIBUTION_IEEE_Center_for_Secure_Design_Release.pdf

Jones, R. L., & Rastogi, A. (2004). Secure coding: building security into the software development life cycle. *Information Systems Security*, 13(5), 29-39.

Microsoft, (2014). *Evolution of the Microsoft SDL*. Retrieved 8 Mar. 2014. <http://www.microsoft.com/security/sdl/resources/evolution.aspx>.

—. *Writing Secure Code*. 8 Mar. 2014. Website. 8 Mar. 2014. <http://msdn.microsoft.com/en-us/security/aa570401.aspx>.

NIAP (2014) *CC/CEM Documentation*. Retrieved 8 Mar. 2014. <https://www.niap-ccevs.org/Documents_and_Guidance/cc_docs.cfm>.

—. *National Policy Regarding the Evaluation of Commercial Products*. 24 Mar. 2005. Retreived 8 March 2014. <https://www.niap-ccevs.org/NIAP_Evolution/faqs/nstissp-11/index.cfm?&CFID=20754558&CFTOKEN=1af2804fc368770e-57C50E2A-B120-B58D-2EF63ACFF8A8F107>.

NIST (2009). *Security Management & Assurance*. Retreived 8 Mar. 2014. <http://csrc.nist.gov/groups/STM/index.html>.

OWASP (2010). OWASP Top 10 Application Security Risks – 2010. Retrieved 3 June 2015. https://www.dropbox.com/home/Secure%20Software%20Development?preview=OWASP_T10_-_2010_rc1.pdf

Patrizio, A. "Home Depot, Target breaches exploited Windows XP flaw, report says." *NetworkWorld. 18* Sep. 2014. *http://www.networkworld.com/article/2685295/microsoft-subnet/home-depot-target-breaches-exploited-windows-xp-flaw-report-says.html*

Paul, M. (2015). White Paper VIII: Software Security in a Flat World. ICS2, Retrieved 3 June 2015. **https://www.isc2.org/csslp-whitepaper.aspx**

Pettigrew, J., Ryan, J., Salous, K., & Mazzuchi, T. (2010). Decision-Making by Effective Information Security Managers. In Proceedings of the 5th International Conference on Information Warfare and Security.

Rashid, Fahmida Y. (2013). Microsoft Talks Secure Coding Practices, *Standards at Security Development Conference*. Retrieved 8 Mar. 2014. <http://www.securityweek.com/microsoft-talks-secure-coding-practices-standards-security-development-conference>.

Richardson, R. (2008). CSI computer crime and security survey. *Computer Security Institute, 1, 1-30.*

Schneier, B. "Heartbleed." *Schneier on Security. 9* Apr. 2014. *https://www.schneier.com/blog/archives/2014/04/heartbleed.html*

Silver-Greenberg, J., Goldstein, M., & Perlroth, N. "JPMorgan Chase Hacking Affects 76 Million Households." *New York Times.* 2 Oct. 2014. *http://dealbook.nytimes.com/2014/10/02/jpmorgan-discovers-further-cyber-security-issues/?_php=true&_type=blogs&_r=1*

Van Wyk, K. R. (2003). Secure coding: principles and practices. *O'Reilly Media, Inc.*".

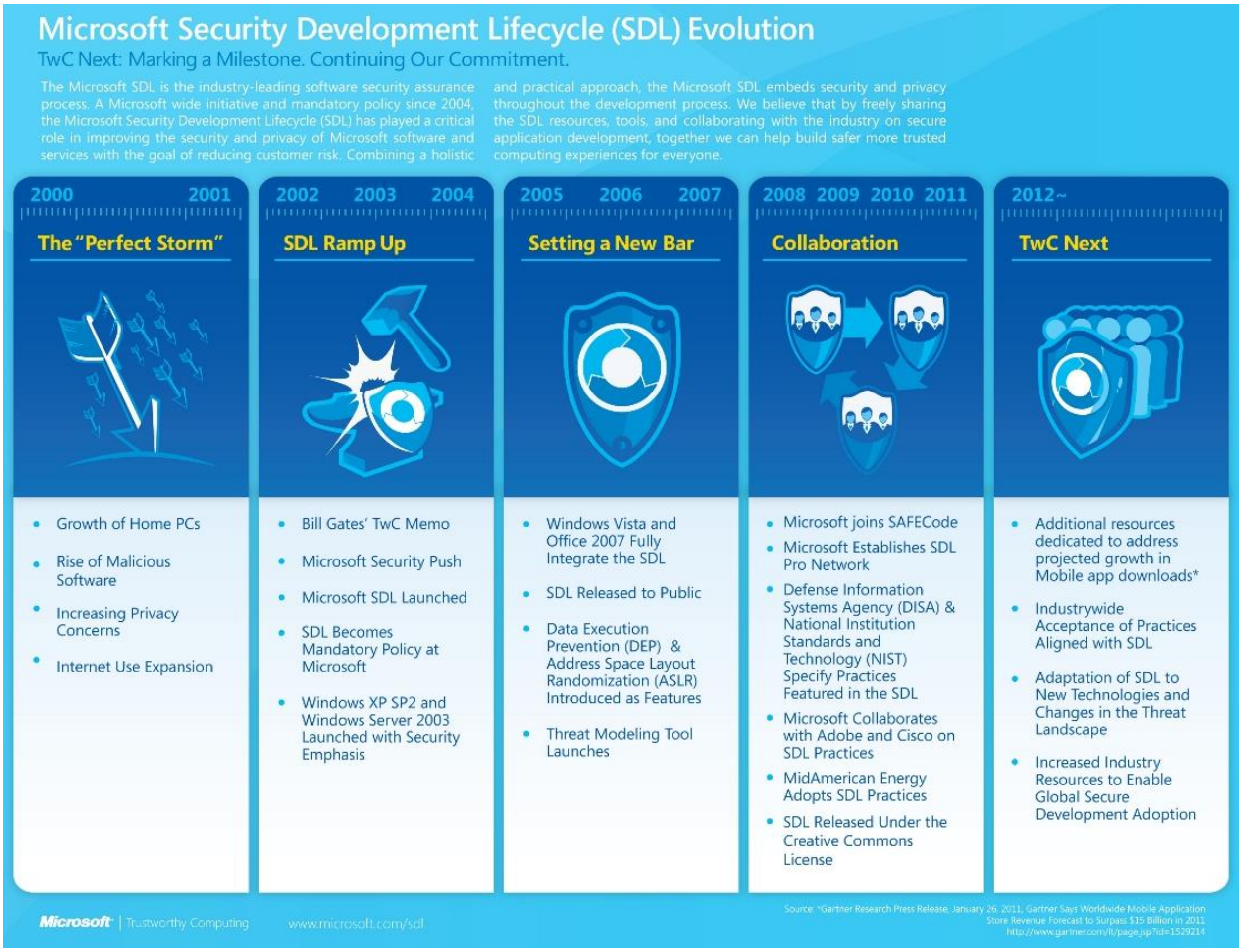
Verizon Solutions, (2014). Verizon 2014 data breach investigations report. http://verizon.com.

Ward, Keith. (2013) Study: Majority of U.S. Developers Use No Secure Coding Processes. Retrieved 8 Mar. 2014. <http://visualstudiomagazine.com/articles/2013/07/16/majority-of-us-devs-dont-practice-secure-coding.aspx>.

Wong, David. (2002) Secure Coding. Retrieved 8 Mar. 2014. <http://www.symantec.com/connect/articles/secure-coding>.

Zafar, Humayun, Myung Ko, and Kweku-Muata Osei-Bryson. (2011) "Does a CIO Matter? Investigating the Impact of IT Security Breaches on Firm Performance Using Tobin's q." *44th Annual Hawaii International Conference on System Sciences (HICSS)*.

Appendix A
Microsoft SDL / Evolution & Timeline



Microsoft Security Development Lifecycle (SDL) Evolution

TwC Next: Marking a Milestone. Continuing Our Commitment.

The Microsoft SDL is the industry-leading software security assurance process. A Microsoft wide initiative and mandatory policy since 2004, the Microsoft Security Development Lifecycle (SDL) has played a critical role in improving the security and privacy of Microsoft software and services with the goal of reducing customer risk. Combining a holistic and practical approach, the Microsoft SDL embeds security and privacy throughout the development process. We believe that by freely sharing the SDL resources, tools, and collaborating with the industry on secure application development, together we can help build safer more trusted computing experiences for everyone.

2000 2001
The "Perfect Storm"

- Growth of Home PCs
- Rise of Malicious Software
- Increasing Privacy Concerns
- Internet Use Expansion

2002 2003 2004
SDL Ramp Up

- Bill Gates' TwC Memo
- Microsoft Security Push
- Microsoft SDL Launched
- SDL Becomes Mandatory Policy at Microsoft
- Windows XP SP2 and Windows Server 2003 Launched with Security Emphasis

2005 2006 2007
Setting a New Bar

- Windows Vista and Office 2007 Fully Integrate the SDL
- SDL Released to Public
- Data Execution Prevention (DEP) & Address Space Layout Randomization (ASLR) Introduced as Features
- Threat Modeling Tool Launches

2008 2009 2010 2011
Collaboration

- Microsoft joins SAFECode
- Microsoft Establishes SDL Pro Network
- Defense Information Systems Agency (DISA) & National Institution Standards and Technology (NIST) Specify Practices Featured in the SDL
- Microsoft Collaborates with Adobe and Cisco on SDL Practices
- MidAmerican Energy Adopts SDL Practices
- SDL Released Under the Creative Commons License

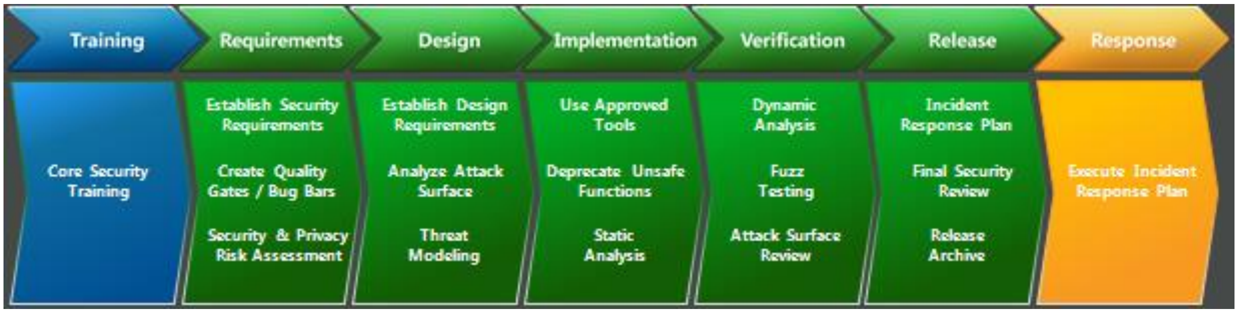
2012~
TwC Next

- Additional resources dedicated to address projected growth in Mobile app downloads*
- Industrywide Acceptance of Practices Aligned with SDL
- Adaptation of SDL to New Technologies and Changes in the Threat Landscape
- Increased Industry Resources to Enable Global Secure Development Adoption

Microsoft | Trustworthy Computing www.microsoft.com/sdl

Source: *Gartner Research Press Release, January 26, 2011, Gartner Says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011 http://www.gartner.com/it/page.jsp?id=1529214

Security Development Lifecycle:



Training	Requirements	Design	Implementation	Verification	Release	Response
Core Security Training	Establish Security Requirements	Establish Design Requirements	Use Approved Tools	Dynamic Analysis	Incident Response Plan	Execute Incident Response Plan
	Create Quality Gates / Bug Bars	Analyze Attack Surface	Deprecate Unsafe Functions	Fuzz Testing	Final Security Review	
	Security & Privacy Risk Assessment	Threat Modeling	Static Analysis	Attack Surface Review	Release Archive	