IJMTES | International Journal of Modern Trends in Engineering and Science
ISSN: 2348-3121

Secure Coding Practices - An Industry Perspective

Milind K. Thombre[1], Dr. Prachi M. Joshi[2]
[1]*(Department of Computer Engineering, MITCOE/SPPU, Pune, India,* thombrem@gmail.com*)*
[2]*(Department of Computer Engineering, MITCOE/SPPU, Pune, India, prachi.joshi@mitcoe.edu.in)*

Abstract—*While there is sufficient study being done in cybersecurity post-priori, there seems to be little being done even today to ensure that badly coded systems do not go to production. Consequently, data is routinely being leaked even by seemingly secure websites. This paper addresses several industry issues around Secure Coding Practices(SCP), why they are not effectively used and the plethora of ignorance on this subject from Software development managers to Project Managers, Business Analysts as well as developers, and testers alike.*

We examine this issue from 3 different perspectives: The System Administrators, Developers/Testers, and Management. We note that there is an urgent need to better educate the entire community involved with software development in SCP.

Keywords—Secure coding practices, Cybersecurity, Cyberdefence, Countermeasures, Cyberwarfare

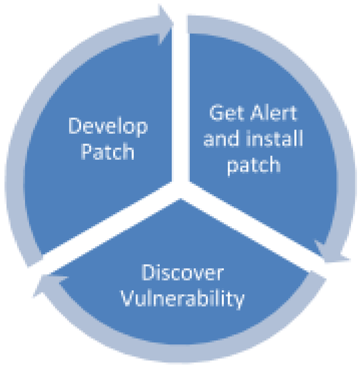
1. Introduction

The topic of cyberwarfare is very broad. Typically, one would start with creating a mind-map of the various subtopics, description of each subtopic, definitions etc. We will however focus on a narrow aspect of this topic namely, "Secure coding Practices".

Any Cyberwarfare activity propagated by a country must begin with strong defense of its own cyber systems. This is necessary to prevent or thwart a counter attack. Cyberwarfare may be initiated not only by 'enemy countries' but also by large international Crime Syndicates and individual hackers working alone.

However, while there is exhaustive literature available on cyberwarfare, there seems to be lesser attention devoted to making computer systems (Application Layer) safer while developing them itself. [1] The developer community is largely unaware of "Secure Coding Practices" even today. With this paper, we will try to address this issue to a certain extent. We must also investigate why the developers are less concerned and educated about defensive coding practices. This issue has many facets. One aspect is that developers are inherently "constructive" people who want to "build" systems and not "break" them. Often the developer will start with creating the system functionality looking at the requirements document itself and tick off each aspect of the requirements as they get developed. Even Business Analysts who write requirements are focused on the question "What should this system do?" and not security aspects that should be incorporated in the system code. Users also are not forthcoming in documenting or elaborating on implicit security requirements beyond the basic authentication and access control. System testers also build test cases to test the system against requirements itself (Requirements traceability matrix) so the security flaws (or lack of definition of security features) in the system propagate to the point when the system is put into production/published.

A. The vulnerability Cycle:



B. System Administrators Perspective:

Any modern system has Several Moving parts or software layers. e.g. in a web application, there is typically an Operating System that hosts a Database Management System (DBMS),Web server/Application server, Development Frameworks etc. Sitting on top of all this, is the code for the web application developed itself. It can be challenging to test such a system from a security angle. A security 'hole' in any layer may compromise the entire system or more importantly leak the user's data that is meant to be secure. Security patches released for **any** of these layers must be applied by the system administrators in a timely fashion so as to keep systems up and safe from hackers. This simple yet effective step is often overlooked by system administrators and Database Administrators. Also, if certain frameworks or infrastructure software stops developing security patches or gets 'de-supported' an alarm must be raised for all systems using this infrastructure software. This must be a part of the system administrators job. A migration plan must be made to migrate applications using this software to competing framework/software and if this is not available, something must be developed quickly to replace it! However, we must note that even patching systems is time consuming and typically involves application downtime. Auto-update utilities must be set ON whenever possible so that systems patch themselves when this option is available. Antivirus and malware removal software must be run daily for each system right from the day the OS is installed. This includes end-points also and not just centralized server systems or cloud based systems. One must not rely on the security system and intrusion detection systems alone and in no case, must one assume that the data is safe even if it is within the enterprise systems and not exposed to the outside world.

B. *The Developer/Tester perspective:*

There has been a large scale "Democratization of development" in Software. [5] While this has its benefits in reducing the cost of software, there is a dark side as well. When developers are less trained in software, they are likely to introduce bugs in the code or not be educated about secure coding practices. This set of less trained developers expose systems to attack. Mandatory training for the developers in SCP is a key before the developers start writing software. This is especially true if they are writing system software. However, web/application software is also extremely vulnerable to attack since it is open on the internet. Often such systems are connected to backend databases which hold critical information. If compromised, this data can fall into wrong hands. These issues seem to be omnipresent, yet are underappreciated in Management circles where the focus remains to somehow reduce the cost of development by introducing inexperienced engineers into the team.

Commercial Static Code analyzers (e.g. HP-Fortify) that can capture several security issues, are still not part of the build cycles of several mission critical software projects. FOSS Static code analyzers (like Pylint)[2] are also not extremely popular even though they have been around for many years. Code review checklists still do not pay enough attention to security features like input checking, output encoding etc. The focus seems to be on functionality like Login/Access control etc. Some developers also have a habit of fixing errors, but ignoring warnings thrown by the compilers. This is potentially a dangerous slip. While schedule and cost pressures are omnipresent, one must complete the task to the best-known standards available.

Here a small discussion about the difference between Quality Assurance and Quality Control is warranted. QA deals with defect prevention, while QC deals with defect detection and fault control. Both these departments must be sufficiently educated about SCP. There is an example of the difference between well engineered software and prototyping in history. The Montgolfier Brothers used rapid prototyping model (parachute discovery). They would throw off an animal in a basket tied to a parachute and then examine the wreckage to further improve the design of the parachute. On the other hand, the Wright Brothers used engineering equations and came up with the airplane which flew on its first attempt. Most Software engineers tend to be like the Montgolfiers instead of the Wrights. We must appreciate that there is value in engineered software, especially good security engineered software as it is extremely disruptive for businesses (e.g. online retail) and loss making to patch a system once it goes to production. Not to mention the risk of losing valuable data.

Testers must be taught about SQL-injection and other attack types which they would otherwise not test for. We must strive to design better test cases covering known security scenarios. Here again training is of paramount importance.

C. *The Management Prespective:*

The Project manager and stakeholders must deal with the reality of competitive pressures. Though they may appreciate SCP, they may not be able to budge for it separately. Then there is also the issue of outsourcing to vendors. Vendors try to outbid each other, typically on Cost. Competition between vendors and lack of insistence on SCP by the Clients will cause havoc later in production systems, when the vendor is long gone from the picture.

The table below shows us that the cost of including security is 67 times more post implementation.

Phase	Cost multiplier
Design	1
Implementation	15
Post implementation	67

D. Review of Secure coding practices

Generally secure coding Practices fall into the following categories [7]

1. Input validation
2. Output encoding
3. Authentication and Password Management.
4. Session Management.
5. Access control
6. Cryptographic Practices
7. Error Handline and Error Logging
8. Data Protection
9. Communication Security
10. System Configuration
11. Database security
12. File Management
13. Memory Management
14. General Coding Practices

E. Example SCP Checklist:[8]

Conduct all data validation on a trusted system (e.g., The server)

Identify all data sources and classify them into trusted and untrusted. Validate all data from untrusted sources (e.g., Databases, file streams, etc.)

There should be a centralized input validation routine for the application

Specify proper character sets, such as UTF-8, for all sources of input and also encode data to a common character set before validating. Determine if the system supports UTF-8 extended character sets and if so, validate after UTF-8 decoding is completed

Verify that header values in both requests and responses contain only ASCII characters

All validation failures should result in input rejection

Validate all client provided data before processing, including all parameters, URLs and HTTP header content (e.g. Cookie names and values). Be sure to include automated post backs from JavaScript, Flash or other embedded code

Validate data from redirects (An attacker may submit malicious content directly to the target of the redirect, thus circumventing application logic and any validation performed before the redirect)

Validate for expected data types

Validate data range and length.

Validate all input against a "white" list of allowed characters, whenever possible

If any potentially hazardous characters must be allowed as input, be sure that you implement additional controls like output encoding, secure task specific APIs and accounting for the utilization of that data throughout the application . Examples of common hazardous characters include: < > " ' % ( ) & + \ \' \"
If your standard validation routine cannot address the following inputs, then they should be checked discretely
Check for null bytes (%00) and new line characters (%0d, %0a, \r, \n)
Check for (../ or ..\) path alterations characters. In cases where UTF-8 extended character set encoding is supported, address alternate representation like: %c0%ae%c0%ae/

2. Conclusion

We conclude that this topic needs to be studied from multiple perspectives and issues need to be examined from multiple angles to reduce the amount and extent of insecure software on the internet.

There is an immediate and growing need to educate developers, testers, PM's, System admins and DBA's as well as sponsors and senior Management about the need to incorporate SCP in the development cycle itself.

References

1. http://www.Owasp.org
2. http://www.pythonsecurity.org/
3. "A study on improving static analysis tools: why are we not using them?" by Brittany Johnson, ICSE' 2012.
4. "Top 10 Secure Coding Practices" by Robert Seacord , SEI-CERT, March 01, 2011
5. Secure Coding: Principles and Practices

   2003, Mark G. Graff, Kenneth R. van Wyk
6. Secure coding Guide, Apple Computer Inc. 2016.
7. William G.J. Halfond, Jeremy Viegas, and Alessandro Orso – "A Classification of SQL Injection Attacks and Countermeasures"
8. "OWASP Secure coding practices Quick reference Guide" 2010.