Improving Software Security During Development

Robert Usher

Submitted to fulfill the practical assignment: SANS Security Essentials GSEC Practical Assignment, Version 1.3

Improving Software Security During Development

Robert W. Usher

March 26, 2002

Abstract

A great deal of work has gone into making computer and network systems secure. This paper will explore the basis for creating secure software and systems during development. Software security directly correlates to the quality of the development process and leadership focus on security. Unfortunately, the market drives demand, so the primary effort is spent on feature rich software rather than secure by default systems. This has made the information security industry what it is today; long checklists to correct default insecurity, expensive audits to discover vulnerable systems and layers of defensive measures to protect against attack.

Software is extremely complex. Each line of code is written by hand and a single system can have large teams of coders all contributing a bit of hand written code. By understanding the concept of Unbounded Systems we can better understand the issues faced by customers and developers. Fortunately there are some methodologies and data that may help improve the quality of systems while they are in development. The Software Engineering Institute (SEI) Capability Maturity Model (CMM), first published in 1993, contains vital practices to improve the software development process. Comparing the CMM with Extreme Programming (XP) we find reinforcement for best practices in development that can lead to better security. A process cannot be altered or controlled unless valid metrics are available. To reinforce the need for change we will examine the results of two studies based on real world data. Evolution of software toward improved security is inevitable, however it will take time.

Penetrate & Patch Securing Unbounded Systems

Security issues tend to focus on network communications. This makes sense since a computer that is not part of a system, communicating in some way, is much less vulnerable to attack. The availability of inexpensive communications (the Internet) has created a requirement for devices to participate in a system of some kind. The rapid adoption of the Internet by business created a need to understand what could happen after joining a public unregulated network. This created the security practice of penetration testing. If the good guys can get in to the system then we know what to fix to prevent the bad guys from gaining the same access.

On the surface this seems like a great idea. Catch the issues before anyone else so that you can fix them. Unfortunately this process spirals out of control because we are working in the realm of the Unbounded System.

According to David Fisher from the SEI: “An unbounded system is any system in which the participants (human or computerized) have only incomplete or imprecise information about the system as a whole. They include human participants as well as automated components.” [1] Consider the assumption about how software will be used. The human input component is impossible to predict. Even a single computer with all the programs required for a specific business can qualify as an Unbounded System. The human input and the interaction of the diverse programs that were never tested together make it impossible to have precise information about the system as a whole.

Once a system connects to the Internet, it becomes a small piece of a larger Unbounded System. The vulnerabilities that are analyzed and corrected to improve security, are actually unpredicted and undesirable states of functionality built into the system. The number of available system states will determine the potential to secure the system.

To secure part of an Unbounded System the system states must be continuously tested: the corollary of penetration testing. When an undesirable state is discovered the state must be eliminated: the corollary of patching. When a patch is applied the system itself is altered and the cycle must be repeated to insure no new undesirable states exist. In addition, any functional change to a system may produce more unknown states that must be accounted for. This never-ending cycle of penetrate and patch is critical to maintaining system security by eliminating vulnerabilities (undesirable states of functionality).

Poor quality leads to security problems just as the unplanned use of well-coded features can. For every change that is made, another test must be used to re-verify the system is functioning correctly. Even with continuous testing there will always be a limit to anticipating the element of human innovation.

Quality is Security

The software economy is driven by customer requirements for new features. Software developers also drive the market when they bring technology innovation to the market first. [2] Software development is done by hand. Although we use words like engineering or science in relation to software and computers, it may be more accurate to think of it as a handcrafted art form. [3] What can be done to create better software and systems?

“Complexity is the worst enemy of security, and systems that are loaded with features, capabilities, and options are much less secure than simple systems that do a few things reliably.” according to Bruce Schneier and Adam Shostack. [4]

This seems like common sense but we should remember that software, being written by hand, could become complex even in a simple form. Particularly when relying on a complex operating system.

The need for mission focused software to reduce the default level of complexity is evident in the checklists that are required to make a system secure. In the SANS Auditing Windows 2000 checklist [5] for example there are 19 steps required to audit the security of Windows 2000. The Auditing Unix [6] checklist has 21 steps and in both cases each step in the checklist may have more than 10 actions or considerations that need to be understood and acted upon. The effort that must be put forth helps demonstrate how increased complexity and flexibility can have a negative impact on security not to mention manageability.

With software that requires extensive configuration management, both for security and day-to-day operations, quality will also be an issue. As the amount of code increases and the number of code writers increase the maturity of the development process must also increase to provide a process capable of producing quality.

Process Creates Quality

Since being published in 1993 the SEI Capability Maturity Model (CMM) has been an excellent guideline for software development organizations. The CMM outlines what to do to create quality software by creating a process and supporting organization. It does not specifically focus on security, however many security issues, like buffer overflows, are a direct result of the quality of the code.

The CMM has 5 levels of increasing maturity that an organization can strive for.

- Level 1 – Initial – Based on the pure ability of people with no formal process or organizational structure. If you write your own programs or scripts, this is you.

- Level 2 – Repeatable – The organization has formal project management, tracking, planning, oversight, and quality assurance.

- Level 3 – Defined – Increased organizational awareness including: process focus and definition, training, integrated software management, product engineering, inter-group coordination and peer reviews.

- Level 4 -- Managed – Metric driven management resulting in quantitative process management and software quality management.

- Level 5 – Optimizing – The inward looking stage of continual improvement including defect prevention, technology change management and process change management.

A process called Extreme Programming (XP) has gained some attention as a better way to create software. Contrasting XP and the CMM, Mark Paulk from the SEI concludes that the CMM and XP are complementary. [7] XP compliments the CMM because it both fulfills many of the specific goals of the CMM while providing a disciplined process to create software. The CMM is a guide on what to do whereas XP is a guide on how to do it.

Comparing the XP process to elements of the CMM we see why XP is complementary to CMM.

XP (Extreme Programming) Process	CMM Goals (Level)
Planning Game: Customer focused release scope planning; includes business priorities/dates and technical estimates.	Project planning (2)
Small Releases: Put simple system into production on two-week release cycle.	Project planning (2) Project tracking (2) Configuration management (2) Defect Prevention (5)
Metaphor: Guide development with shared story of system functionality.	Requirements management (2) Project tracking (2) Project engineering (3)
Simple Design: smallest, simplest code possible to do the job.	Project engineering (3) Defect Prevention (5)
Testing: Continuous unit testing. Unit test is done prior to coding.	Project engineering (3) Defect Prevention (5)
Refactoring: Restructure system to remove duplication and simplify.	Project engineering (3)
Pair Programming: All code written by two programmers at one machine.	Quality assurance (2) Inter-group coordination (3) Peer reviews (3)
Collective Ownership: Anyone can improve any code at any time.	Configuration management (2)
Continuous Integration: Code is integrated at least every day. Continual regression testing of system.	Requirements management (2) Configuration management (2) Project engineering (3)
40-hour Week: Work no more than 40 hours per week. Never work overtime two weeks in a row.	
Onsite Customer: An actual customer is onsite full time to answer questions.	Requirements management (2) Inter-group coordination (3)
Coding Standards: Agree on rules that emphasize communication throughout the code.	Project engineering (3)

The XP process meets many of the CMM goals for Levels 2 and 3. Use of XP would not hinder and organization from attaining the higher CMM levels. We should be interested in how the critical software we depend on is created and the process that is used. An increased understanding of the software development process will help refine perceptions about secure software.

Although the combination of the CMM and XP may help an organization create better software, supporting data is required to provide evidence that these processes need to view security as a quality issue. Supporting data can be found in the Hoover Project and SEI research on Survivability. Both quality and complexity of software design are tied together in many ways. Increased complexity may lead to lower security due to pure quality issues, increased number of system states and overly complex configuration scenarios. With real data, realistic decisions can be made.

Hoover Project

Taken from 18 months of e-commerce consulting engagements the Hoover Project compiled a vast amount of customer data to discover differences in software development techniques. The best-designed software produced 82% less risk than the worst. [8] The differentiating factors and recommended actions that resulted from the best/worst analysis clearly show security can be mitigated during development.

Differentiating Factors

- Early design focus on user authentication and authorization – Similar to the SANS/FBI Top 20 topic, “Accounts with no passwords or weak passwords” [9], authentication remains one of the most important security issues. The development of customized e-commerce systems requires as much attention as the authentication security for an operating system. The development team can use different types of authentication databases and techniques. The most important factors that created the best results were password encryption and a strong authentication database. The best implementations reduced business risk by 88% vs. the worst.

- Mistrust of user input – Initially this seems very specific to e-commerce where input could contain code, like html, with a malicious purpose. This point can be extrapolated to all software when we consider that human input cannot be predicted. Increasing the stability of software systems by adding input protection decreased business risk by 77%.

- End-to-end session encryption – To produce a 90% business risk reduction, session encryption is critical. Without proper session management and encryption, applications are susceptible to session

hijacking where a replay of the current session identifier can give the attacker the privilege of the user.

- Safe data handling – Similar to the data security required while data is on the network, data must be also be encrypted and segmented wherever it is stored. Nothing is more embarrassing than customers seeing each other's data! Focus on the use of proven encryption reduced business risk by 93%.

- Elimination of administrator backdoors, miss-configurations and default settings – The SANS/FBI Top 20 lists "Default installs of operating systems and applications" [9] is at the top of the list. Following this best practice reduced business risk by 82%.

- Security quality assurance – The Hoover Project did not report any hard data on this, but does note that making security part of the quality assurance process helped the high achievers create the better applications. Also noted is the reduced post-production impact due to an emphasis on security within the development team.

Although this data was taken from e-commerce projects its applicability is more general. Combined with other sources, like the SANS/FBI Top 20, software developers should be able to justify building security into products early. The actions that the Hoover Project concluded are also useful for development teams and customers to consider.

Actions

- Stop depending on the firewall – Certainly good advice if you want to think about your true security posture. Without the firewall we would all be in trouble, however the firewall is still going to allow some access to your systems. That access may lead to a targeted application attack that the application must be able to withstand without failing. If systems are designed to with stand attack, continue their mission, and recover then the firewall will be obsolete. Attaining this level of Survivability [10] is going to take some time.

- Act up – This guidance from @stake is addressing the fact that customers must provide feedback to software developers. This is a market issue. Customers can increase their knowledge about a software purchase by educating themselves and asking appropriate questions. The best feedback is revenue. Cash flow can fund better security or complex features. Expenditures will convey the desire of the market, as long as the market has complete access to the pertinent information.

- Educate application developers/Engage Finance and Audit – Focus on the root causes during development and let the boardroom create the security focus through leadership. The entire weight of the issue cannot be dropped on the developers alone. Looking at the CMM, XP, and other quality management programs like ISO 9000 and TQM one point is clear: Leadership makes the difference. Quality and security are not going to change until the change is made at the top.

- Assess early and often – Avoid post-production surprises with constant assessment during development. Penetration testing will become part of development. The XP programming process has 3 places for ongoing assessment: Developer Unit Testing, Integration Testing, and Coding Rules. By focusing the XP process on security the process will drive out security defects.

- Get outside help – Security skills are rare. Seeking out the expertise that exists outside the organization can enhance understanding secure software development. Outside help is also available in written form. Read everything you can to make better decisions.

Survivability

The SEI research on systems survivability has provided an empirical study based on the reported CERT incidents from 1988 to 1995. [10] The modeling of this real data is a positive way to create better understanding of the need for system security. In addition a cost/benefit analysis was preformed to determine the effect of increased security spending. Unlike the Hoover Project that focused on security during development, this study examines security based on configuration and external defenses.

Cost is the aggregate of additional defensive measures (like firewalls) and potential reduction of desired functionality (like disabling a useful feature) in trade for a more robust security posture.

Survivability is the ability of a system to maintain a functional (although possibly compromised) state that still fulfills the mission of the system. An example would be a mail server that still sends mail even though the local web service has been crashed.

Survivability Research Summary

Damage is directly proportional to the rate of attack. The faster the attacks occur the more damage will result. This is still the case even when the attacks are not sophisticated, however the number of weaker attacks over time must be greater.

Survivability is inversely proportional to the number of available system states. The more states a system can have the less likely it is to survive under attack. This reinforces the early statement that complexity is the enemy of security. It also supports the XP process elements: Simple Design and Refactoring that help eliminate superfluous code.
Survivability is not as affected by attack strength or attack rate. If there are few available system states, the attacks have a minimal chance of success. If the system has 1000's of different functional states there will as many successful attacks discovered for that system.

Cost of Survivability is a curve. Initial spending creates a dramatic increase in Survivability. At some point the curve starts to flatten and a similar increase in spending will now produce only a quarter of the initial result. The cost figure is theoretical, but indicates an optimal level of expenditure on defensive measures.

Common Criteria

As an example of additional sources of input to improve the development of secure software we could incorporate concepts from the Common Criteria Evaluation Assurance Levels (EAL) into the CMM/XP process. Common Criteria evaluations are based primarily on Protection Profiles and Security Targets.

Protection Profiles, simply stated, are written by a customer to define the security requirements for a product. [11] To inject the Protection Profile into the combined CMM/XP process we look to the XP element of the Onsite Customer. Giving the Onsite Customer guidelines for planning the security requirements will help produce more secure software.

Security Targets, on the other hand, are written by the developers to detail how the software will meet the Protection Profile. [11] This fits well into XP test plans and developer unit testing. Both the Protection Profile and the Security Target requirements would dictate the plan in Continuous Integration testing phase.

Even if the development process does not require a formal evaluation under the Common Criteria, further understanding by developers and managers can help promote a security conscious development organization.

Conclusion

Quality and complexity both have a role in the security of software and systems. The software development process can be guided by the CMM and XP when taken in the context of real historical data. Improving the process while understanding the data should produce better, more secure, software and systems. The development lifecycle process will change slowly. Thus the current security practice will continue to evolve. Everyone who touches software or systems must make a commitment to improve the state of security.

Understanding the development process and the inherent challenges are part of the cycle of continuous improvement.

Future study

With the vast amount of research and activity underway to improve system quality and security, the need for continued research is evident. A correlation of the SEI Survivability data with the Hoover Project to understand the impact of proactive application security vs. external defenses would be valuable. Validation of the development process through analysis of additional real world data will be critical. Whatever data is gathered should always be used as a reference point to guide the process. Software development organizations could analyze and publish security data in support of specific process elements. Although competition might prevent such disclosure, academic research may be able to attain the same goal through long-term corporate engagements. A topic not covered here is the legal aspect of software and system security. There are issues that may impact quality and security found in current legislation, which may be worthy of future exploration. [12]

[1] Fisher, David A. “Survivability and Simulation”, Software Engineering Institute, November 14, 2000
URL: http://www.cert.org/easel/survsimesl.html

[2] Baskerville, Levine, Pries-Heje, Ramesh and Slaughter, “How Internet Software Companies Negotiate Quality”, May 2001, IEEE Computer
URL: http://computer.org/computer/homepage/may/baskerville/index.htm

[3] Hayes, Will and Over, James W. “The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers “December 1997
URL: http://www.sei.cmu.edu/pub/documents/97.reports/pdf/97tr001.pdf

[4] Schneier, Bruce and Shostack, Adam. “Results, Not Resolutions, A guide to judging Microsoft's security progress.” January 24, 2002
URL: http://online.securityfocus.com/news/315

[5] SANS, “Auditing Winodws 2000”
URL: http://www.sans.org/SCORE/checklists/AuditingWindowsNT.doc

[6] SANS, “Auditing Unix (Solaris)”
URL: http://www.sans.org/SCORE/checklists/AuditingUnix.doc

[7] Paulk, Mark C. “Extreme Programming from a CMM Perspective” IEEE Software November/December 2001
URL: http://www.sei.cmu.edu/cmm/papers/xp-cmm.pdf

[8] Jaquith, Andrew. “The Security of Applications: Not All Are Created Equal” February 2002
URL: http://www.atstake.com/research/reports/atstake_app_unequal.pdf

[9] SANS/FBI Top 20 – “The Twenty Most Critical Internet Security Vulnerabilities” Version 2.502 January 30, 2002
URL: http://www.sans.org/top20.htm

[10] Konda, Moitra. “The Survivability of Network Systems An Empirical Analysis” December 2000
URL: http://www.cert.org/archive/pdf/00tr021.pdf

[11] Aizuddin, Ariffuddin. “The Common Criteria ISO/IEC 15408– The Insight, Some Thoughts, Questions and Issues” October 1, 2001
URL:http://rr.sans.org/standards/ISOIEC_15408.php

[12] Braucher, Jean and Henerson, Roger. “The Uniform Computer Information Transactions Act (UCITA): Objections From The Consumer Perspective” August 21, 2000
URL: http://www.cpsr.org/program/UCITA/braucher.rtf