Global Information Assurance Certification Paper

A Practical Guide for Secure Software Development and Quality Assurance

Jong C. Hong
Sept. 11, 2002
GSEC Practical v1.4 option 1

Abstract – Today software is everywhere – cellular phone, home PC, enterprise applications, networking device, firewall, and e-commerce just to name a few. As the software industry grows so does the risk of an incident with software security. Its effect can be global in scope and the loss can be monetary or even human life. All because of the defective software, cracker, weekend warrior and script kiddies alike are using automated hacking tools [1] ready to do harm for fun or recognition, corporate espionage steals intellectual property for profit gain and terrorist can attack the infrastructure of a nation as a whole [2]. And yet the software vendors remain defiant and are producing defective products every business day. To reduce the risk that we all are bearing, managers at all levels of a software company should fully realize the seriousness of making insecure software and therefore institute proper policy to minimize future incidents. Software developers should raise their awareness of security issues during the development process and produce security-proof software.

There are already many articles talking about secure software [3] [4] and it is not the intention of this paper to be Yet Another Secure Software (YASS). However, it appears to the author that a majority of today's software developers still have a perception gap between programming and security. To bridge this gap, this paper tries to give developers a reality check with the latest security incidents while discussing secure software in principle based on the author's experience as a software practitioner. It also intends to serve as a placeholder for readers who are searching for materials and knowledge for developing secure software.

This paper assumes the reader has some knowledge of operating systems, software and security in general. Its audience can be a manager, system architect, security consultant, or programmer. It begins with analyzing existing issues within the software industry in terms of security and identifying typical defects and security attacks. The concept of risk assessment is then introduced to provide a way of thinking for developing software with security perspective. Lastly, guidelines for implementing secure software are discussed along with software development life cycle.

Software Security – As software grows as a big business, a brand new software security industry is arising. If you search "software security" with an Internet search engine you will be amazed at the number of pages talking about it. Generally speaking software security deals with confidentiality, integrity and availability aspects of a software product. Before discussing software security it will be helpful to take a look at what has made software security become a business.

- Software has become more complex and difficult to manage. Quoted from LA Times "Intel employs more than 5000 tech supporters. Much of this is due to software complexity, program bugs, and poor quality in software programming. These all add up to an immense burden on the economy" [5]. This is the major reason why there are so many holes created in the process of making software products.

- More and more valuable information is carried via network software. Internet, Intranet and Extranet all are part of modern business. Any unintended mistake creates a chance for information leakage.

- Pressure from tight schedules and budgets often compromise security issues. Management is often shortsighted about the potential penalty of creating insecure software.

- Lack of governmental standard and auditing agency. NIST Computer Security Division (CSD) [6] has taken the lead to address this issue. It provides several guidelines and publications for governmental and private businesses to follow for security concerns. Also the health care industry now has its own HIPAA [7] initiative to deal with security issues.

- Legacy of a closed system mentality creates a closed-door policy. An open source platform is much more secure than a close system because anyone can see the insides of an open system and security holes get spotted and fixed quickly. Not so for a close system.

- None or minimum monetary responsibility for creating a defective product. Future legislation can give a wake-up call to all software companies.

- Revision, service pack, and hot fix become publicly acceptable behavior and ironically they are even rewarded as moneymakers.

- The development community lacks the awareness of security issues (i.e. exploit and abuse). Most programmers probably have no idea that a software crash can lead to a possible exploitation. Or a crafted message can let a hacker take control of the entire system.

Software Defects – The bug list of Microsoft W2K SP2 [8] gives us a glance of some possible sources of software defects. To choose this list as an example does not qualify other systems as better products but just its availability. It helps to place a real face on software bugs in terms of security so that programmers can better recognize them in their own programming life and hopefully avoid making the same mistakes. We are going to cluster and examine some of these representative bugs.

1. Design Flaw – missing security requirements or functionality that designers did not recognize their importance or did not plan for them in advance. This is the most profound mistake and takes a longer cycle to fix. Usually it is something that has impact on the entire system or a subsystem.

   - Q258872 – *"Error Code 1350 Applying SetFileSecurity() to COMx".*

     -> It will no allow set security on COMx. Quite often a newly added feature, in this case the security, is incompatible or missing for the older system, which still need to be supported. If security had been considered when the original system was designed, it would be much easier for adapting later.

   - Q262979 – *"Cannot Renew Verisign Certificates in IIS 5.0".*

     -> It only recognizes PKCS#7 but not PKCS#10. The original design did not make room in advance for adapting future technology or standard.

   - Q269239 – *"Vulnerability May Cause Duplicate Name on the Network Conflicts", "The NetBIOS over TCP/IP (NBT) protocols are, by design, unauthenticated and therefore vulnerable to "spoofing." "*

     -> A protocol without capability for authentication makes spoofing possible. Why not enable protocols to be authenticable during design phase even you don't need to implement immediately.

   - *Q266794* – *"Windows 2000 SNMP Registry Entries Are Saved in Plain Text Format and Are Readable".*

     -> Somehow designer has to think like a hacker. SNMP is designed for network administrators but hackers also can use it for malicious act.

2. Logic Fault – In your mind you think it makes sense but actually it doesn't and consequently creates a potential security hole. This fault takes all sorts of shapes and forms. It can be a mistaken logic, timing issue [9], or memory leak etc.

   - Q262539 – *"Memory Leak in Lsass.exe with Large Built-in Groups"*

     -> Forgot to release memory when it is no longer in use and it can be targeted for a deny of service attack.

   - Q263603 – *"Incorrect Behavior in Winlogon for First-Time User with "Must Change Password on First Logon" Setting"*

- Q263743 – *"RasDisable and RasForce WinLogon Policies Can Be Bypassed"– "The code to expand the dialog box does not check the settings for RasDisable or RasForce before enabling the check box"*

  -> Both are possibly caused by mistaken logic for some boundary cases. Mistaken logic can create possible exploitation limited only by a hacker's imagination.

- Q289166 – *"Race Condition Occurs and Autochk.exe Stops Responding During Restart"*

  -> A race condition can create a small window for exploitation such as illegal file accesses or produce unpredictable system behavior. It is more difficult to trace if it is between software and hardware interface due to its tiny window.

3. Incomplete Data Coverage – it is a kind of logic fault but significant enough to be a separate category. It is that something you know you should do with the data but you didn't. Either you think it is a case that will never happen, or you promise that you will do it later when you have time, or you assume somebody else will do it. The later one typically involves an interface issue.

   - Q286132 – *"does not correctly handle a particular series of data packets and cause blue screen."*

   - Q278499 - *"This vulnerability, known as Cross-Site Scripting (CSS), results when web applications don't properly validate inputs before using them in dynamic web pages."*

   - Q267843 - *"Windows 2000 Telnet Server Stops Responding After Binary Input"*

   - Q274835 – *"Buffer Overflow in Network Monitor May Cause Vulnerability"*

     -> All the above are candidates for deny of service attack.

4. Others - simple typo that generates unintended code; compiler generates erroneous code; integrates with faulty library functions or mis-configuration.

Types of Attack – There are plenty of web sites that discuss security breach methods [10] [11]**.** Here we will not get into detail but collect links for some typical attacks for the reader's further study. The purpose is to provide background information for later discussion and also let the reader familiarize themselves with terminology and typical hacking schemes. Most links also provide an associated incident for understanding the nature of the attack.

- Deny Of Service attack (DOS) - http://www.cert.org/tech_tips/denial_of_service.html

- Distributed Deny Of Service attack (DDOS) - http://www.cisco.com/warp/public/707/newsflash.html

- Malicious Code attack -
  - Trojan horse - http://www.cert.org/advisories/CA-1999-02.html
  - Virus - http://www.pcworld.com/features/article/0,aid,31002,00.asp

- Sniffing (Eavesdropping) - http://www.robertgraham.com/pubs/sniffing-faq.html

- Spoofing **-** http://www.itsecurity.com/papers/articsoft9.htm

- Privilege Escalation - http://www.windowsitsecurity.com/Articles/Index.cfm?ArticleID=9195

- Cryptanalytic (Brutal force) attack (e.g. password) – http://www.rsasecurity.com/rsalabs/faq/2-4-2.html

- Session Hijacking - http://www.owasp.org/asac/auth-session/hijack.shtml

Risk Assessment – When your customers purchase insecure software from your company, they risk revenue loss, lost productivity, liability costs, and brand damage [12]. All that can become your liability. On the other hand, you know software development is a daunting task and requires resources, which seems to always reach a limit. All in all there is no absolute secure software! So ultimately you have to ask at what cost to achieve what level of security? To help to answer the question, Tim Bass and Roger Robichaux's paper suggests doing risk analysis by evaluating three security risk elements - criticality, threat and vulnerability, with a Risk-Qualifying Matrix [13]. Although the paper is meant for general IT security, the principle can be applied here for risk assessment on producing secure software. Basically you have to know –

- Where are the critical components in your system? For example, the user password or system configuration file is a critical asset; a random number generator referenced by 20 other modules is definitely crucial to the system.

- What are the threads to these components? Internet exposure or untrustworthy user access is certainly a potential thread.

- What and how components are vulnerable to the thread? A component that accepts user input data is vulnerable to buffer overflow exploitation. The component allowed to change privilege is a target for privilege escalation.

At the end of exercising the risk assessment you should be able to arrive at the most cost-effective development process that fits your business. In other word, with given resources (people, budge and time), you should be able to prioritize your resource allocation based on projected risk factors.

Guideline for Secure Software – Software development is in a cycle. For each stage of the software development, implementers should examine every aspect of security issues. Note that it requires a sound plan and discipline to produce secure software, as we will discuss in this section.

Functional spec – this phase creates a blueprint that specifies what services the software will provide [14]. It is the contract between developers and their clients. The author of this document should make security a requirement and request a definite plan to implement and verify it during entire development cycle.

- It should evaluate the necessity of security features for the system – such as secure link between nodes, system-wide key management etc.
- It should detail possibilities of abuse of the system – database tampering, deny of service, spoofing, or brutal force attack etc.
- It should specify countermeasure along with recovery scheme.

High-level design – this phase specifying the overall software architecture and breaking downs the system into subsystem modules using a software design methodology, either Object-Oriented Design [15] or a traditional Structured Design approach by Yourdon [16]. You should always consider using design methodology with your project. "The availability of significant measures in the early phases of the software development life-cycle allows for better management of the later phases, and more effective quality assessment when quality can be more easily affected by preventive or corrective actions." [17]
The following list provides some guidelines during high-level design phase within the security domain.

- Design in security – always keep security in mind and ask questions regarding Confidentiality, Availability and Integrity, such as where is the vulnerability? Where should data be encrypted? How users should be authenticated? Where and how system can be exploited? When you draw a security perimeter, don't forget a perpetrator can be either an outsider or an insider.

- Defense in depth – install security gates at multi-layer with multi-tier. It could be from applications to OS and from user interface to database access. It is based on the concept of deterrence instead of a single knockout.

- Accurate protocol - note that poor design leads to security holes. The TCP/IP stack is not secure by design [18] because of lack of authentication. The 802.11 WLAN protocol is not secure [19] due to several known vulnerability. Traditionally performance and functionality is the centerpiece of the protocol design and nowadays security should be on the design list.

- Completeness - cover up all the holes as much as possible: e.g. there are 6 TCP flag bits so how many cases do you actually consider? If possible verifying your design with simulation or fast-prototype before putting it in action.

- Traceability - logging is your final defense in detection and recovery. If every security measure failed you want to know what did go wrong and how to fix it. But beware the possible exploitation on the log itself. Log information has to be concise and sufficient.

- Strongest password - installs strongest password policy and enforces it. For a cracker there are tools ready for brutal force cracking and why should you let them have the easy way?

- Weakest link - know where your weakest link is. Usually it is not as obvious as hash algorithm, but as simple as where and how your key is stored.

- Default case - default to a case not because it is the most common one but because it is the most security one. For example, set router default configuration to no remote management is allowed as oppose to worldwide accessible.

- Encryption - use the strongest encryption algorithm available.

- Paranoid is virtue - pay attention to every little thing. When backing up or restoring the key do you require a password?

- Legal banner – place a legal banner on login screen to deter intruders. Consult legal counsel for the content of the legal banner.

- Security API - examine and harden security features of the underlying platform. Whether it is OS, database or any other subsystems you build upon, make sure you understand its API of security features and use them properly. For instance, if you build your application on top of Windows you should know the right way to set security or control access levels [20].

Low-level design – with this phase, the internal logic and data of each of the internal processes is defined. The deliverable results in a low-level design document will be the guidebook for the programmer.

- Clear interface – try to define clean interface between modules. Confusion does create mistakes. People tend to be reluctant to change their code once it is done, which can lead to sloppy modifications. It usually takes more effort to modify code than to make it right in the first place.

- Memory management – memory is a limited resource and can be exploited. Design a graceful recovery from memory exhaustion. Beware of memory leaks and provide a way to detect them. Always consider where to release it at the time you allocate it.

- Random number generation (RNG) – the goal is to make it unpredictable. There are several sites for RNG and testing[21] [22]. Carefully select one and make it available to every module that needs it. Generally speaking, hardware based RNG (i.e. Intel 810) is better than software based RNG.

- Debugging tool – well designed system-wide debugging tools can alleviate team members' debugging effort and improve quality of code. Such as memory usage monitor, or event log.

- Input validation – require that all user input is checked and double-checked.

Coding and Unit testing – with this phase, module definition of low-level design is converted into computer executable code with chosen programming language. Each individual programmer himself tests the executable code against potential errors. Note coding is only half the task and the other half is to make sure of its correctness. Here are some needs to pay attention to.

- Top-down coding, bottom-up testing - by that it means that starting your coding from a broader view and don't let the nitty gritty bog you down at the very beginning of your coding task. And testing your code starts from smallest module as you can.

- Border case - be careful for border checking and corner case handling. What is your assumption? And can your assumption stand in the real world? Above all be sure to add assertions (exception handling) for any potential corner case concerns.

- Dangling pointers - a reference that doesn't lead to anywhere. This happens because it formerly pointed to something that has moved or disappeared. It is a very subtle programming bug and is very difficult to isolate. In the C world, in order to avoid this bug, always set a pointer to 0, when delete is called. Subsequent attempts to use the pointer will result in a run-time exception. This will immediately allow the bug to be identified and fixed. In the C++ world a smart pointer is the way to go. [23] [24]

- Buffer overflow exploitation – Statistics show that the majority of today's security breaches originate from buffer overflow exploitation. Technically it is caused by a lack of bounds checking on data buffers. Programmers need to fully understand how it works and how it is exploited. [25]

- Data type conversion – It is done by either casting to force the compiler to use a different data type, or by implicit conversion rules. Be careful of the side effect of data type conversion. These are two security advisories regarding data type conversion [26] [27].

- Race condition – More than one process is allowed when competing for the same resource. You need to either serialize the access or avoid it altogether. Otherwise, unexpected consequences will occur and can be exploitable[28].

- Code review – Before checking in source code it should be reviewed by a team to verify the correctness of the code and to identify possible security vulnerabilities. The reviewee should understand that defects are normal even for the most skilled programmer - don't be unreasonably defensive. The reviewers should make an effort to study the code before attending the session – ask intelligent questions.

- Documentation – good documentation prevents misunderstanding and improves code quality and maintainability.

Integration – with this phase all source codes are combined and tested as a whole system.

- Source code control - Use the source code control tool to automate the integration process and version control. It helps to reduce human error and makes it easier to trace.

- Code check-in - Can your integration process be tampered? Make controllable check in so you know who checked in what. Code should be protected by signature.

- Third party code - Is your third party code secure? How trustworthy is your third party code? Your third party code can come from any source including your enemy. Be ware that today's software comes from every corner on earth.

Quality Assurance – a process to validate software product for required functionality, standard and quality. Here is an official site that provides information on QA - http://satc.gsfc.nasa.gov/fi/fipage.html. I will add the following.

- Testing tools - Beside your own test tools, there are free hacking tool kits out there for you to test your software. However you have to be very careful about the source of your free software.

- Penetration test – do your own penetration test, a controlled network attack simulation. It identifies specific exploitable vulnerabilities and risks within your software.

- Regression test - You might fix a bug but create 10 others [29]. Give time toward testing your patch before releasing it. It is a double jeopardy to make double fouls.

Software Release – the software is released to general public.

- Default configurations - always set default configuration with secure concern. This one is on the top of SANS/FBI's "The Twenty Most Critical Internet Security Vulnerabilities List" [30].

- Code tampering – be ware someone will tamper with your distribution code.

Conclusion – Information warfare is like an arms race that constantly escalated by opposing party. Computer technology and malicious threat are changing every day. A company can only survive by sensing the need from its customer. Security is definitely on the top of the requirement list of your clients for the years to come. Therefore educate your people to be secure minded.

Yes, by the way, be sure your software development environment is secure! Somebody might be watching your work.

[1] Hacking Tools Links - hackingexposed.com - URL: http://www.hackingexposed.com/tools/tools.html

[2] Vamosi, Robert. "Cyberterrorists don't care about your PC", ZDNet Reviews, July 10, 2002, 4:35 AM PT, URL: http://zdnet.com.com/2100-1107-942701

[3] Stokely, Murray. "Chapter 3 Secure Programming", FreeBSD Developers' Handbook, URL: http://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/secure.html

[4] Wheeler, David A. "Secure Programming for Linux and Unix HOWTO", v2.966, 13 July 2002 URL: http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html

[5] Cochran, Shannon. "The Rising Costs of Software Complexity", *Dr. Dobb's Journal* April 2001, URL: http://www.ddj.com/documents/s=868/ddj0104n/0104n.htm

[6] NIST Computer Security Division, URL: http://csrc.nist.gov/

[7] The Health Insurance Portability and Accountability Act of 1996 (HIPAA), URL: http://cms.hhs.gov/hipaa/

[8] List of Bugs Fixed in Windows 2000 Service Pack 2 (1 of 4), URL: http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q282522&

[9] Stokely, Murray. "Chapter 3 Secure Programming 3.7 Race Conditions", FreeBSD Developers' Handbook, URL: http://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/secure-race-conditions.html

[10] "… Learning About Security Breach Methods", Saturday 24 Aug, 2002, Broadband-Help, URL: http://www.broadband-help.com/cm_security.asp#3

[11] "The Open Web Application Security Project", URL: http://www.owasp.org/asac/

[12] DASH, JULEKHA, "Eli Lilly cites programming error for e-mail privacy gaffe", JULY 05, 2000, ComputerWorld, URL: http://www.computerworld.com/securitytopics/security/privacy/story/0,10801,61934,00.html

[13] Bass, Tim & Robichaux, Roger. "DEFENSE-IN-DEPTH REVISITED: QUALITATIVE RISK ANALYSIS METHODOLOGY FOR COMPLEX NETWORK-CENTRIC OPERATIONS", URL: http://www.silkroad.com/papers/pdf/00985765.pdf

[14] "CSE 476/486 - Senior Capstone Design Functional Specification Guidelines", http://www.cet.nau.edu/~edo/Classes/CSE486_WWW/Docs/Guidelines/fn_spec_gdlns.html

[15] Graham, Ian. & Wills, Alan. "ULM Tutorial", URL: http://uml.tutorials.trireme.com/

[16] Yourdon, E. & Constantine, L.L. "Structured Design". Englewood Cliffs, NJ: Prentice Hall, 1979

[17] Lionel C. Briand, Sandro Morasca, Victor R. Basili, "Defining and Validating Measures for Object-Based High-Level Design", **URL**: http://citeseer.nj.nec.com/514589.html

[18] Bellovin S.M. "Security Problems in the TCP/IP Protocol Suite", URL: http://www.ja.net/CERT/Bellovin/TCP-IP_Security_Problems.html

[19] Karygiannis,Tom & Owens, Les. "Wireless Network Security 802.11, Bluetooth™ and Handheld Devices" NIST Special Publication 800-48, URL: http://csrc.nist.gov/publications/drafts/draft-sp800-48.pdf

[20] Articles about Writing Secure Code with Windows, URL: http://www.windowsitsecurity.com/Articles/Index.cfm?DepartmentID=753

[21] "A STATISTICAL TEST SUITE FOR RANDOM AND PSEUDORANDOM NUMBER GENERATORS FOR CRYPTOGRAPHIC APPLICATIONS", URL: http://csrc.nist.gov/rng/SP800-22b.pd

[22] Seifried, Kurt. "Why Random Numbers Are Important For Security", January 26, 2000 URL:http://www.seifried.org/security/cryptography/20000126-random-numbers.html

[23] http://ootips.org/yonat/4dev/smart-pointers.html

[24] http://portal.acm.org/citation.cfm?id=176454.176504&coll=portal&dl=ACM&idx=J513&part=journal&WantType=Journals&title=LOPLAS

[25] http://nsfsecurity.pr.erau.edu/bom/

[26] Starzetz, Paul. "Quick Analyses of the recent crc32 ssh(d) bug", URL: http://reactor-core.org/security/integer-overflow.html

[27] "Microsoft Security Bulletin (MS99-045)", URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/fq99-045.asp

[28] http://razor.bindview.com/publish/advisories/adv_chfn.html

[29] Thomas, Lillian. "911 system failure tied to computer programming error" Wednesday, May 01, 2002, Post-Gazette http://www.post-gazette.com/localnews/20020501911out5.asp

[30] The SANS Institute, Version 2.504 May 2, 2002. URL: http://www.sans.org/top20.htm