Building a secure system: Why do we need to do it? How do we get it? And where should we start?

CGI

Paul Hopkins
Cyber Security
Technical Authority
paul.hopkins@cgi.com

Wendy Holt
Strategy &
Innovation Director
wendy.holt@cgi.com

Extract from 'Cyber Risk: Resources for Practitioners' published by the Institute of Risk Management 2014.

Building a secure system: Why do we need to do it? How do we get it? And where should we start?

Paul Hopkins, CGI

Abstract:

Whether an organisation builds software itself, integrates third parties or just procures a solution, the risks of a poorly secured system will ultimately have a significant impact on the business. Secure systems engineering, is not something that usually gets prioritised against the need to get to market quickly or reduce costs, unless of course the business truly understands the risks.

This chapter covers the risks arising from poor security engineering. We look at the potential impact on the business, what steps can be taken to mitigate these and therefore what questions all risk managers should be asking their internal and external architects and developers, highlighting the potential impact of software development methods such as AGILE on secure development methodologies.

Insecure Systems – Bad for business?

You only have to read about the increasing number of publicly quoted data breaches to see that an insecure system is not good for business. If you fail to keep your customers', or your organisation's, data confidential and available then potentially the consequences could be serious. The organisation could face a huge fine (e.g. TJ Maxx [8]), it could lose customers, it could get swallowed up by the competition (e.g. HBGary [7]) or it could contribute to the Company going out of business (e.g. Nortel [9]).

Not all of the publicly quoted examples above are just the result of an insecure system design and build, indeed social engineering of employees (alongside other factors) contributed to the end result. However, at the end of the day, the systems were compromised and exploited due to weaknesses in the overall system design that were not adequately assessed at the outset by the system designers.

It is possible you may be fortunate enough not to have your business put at risk as a consequence of a security breach. However, it is likely that the cost and difficulty of fixing the issues once discovered are substantially more than identifying and mitigating the issues earlier in the system development process. For a start, the problem may be within a third party component and it might not be possible to fix it directly yourself without their help. Alternatively the flaw may be in your software or technology so you might have the ability to directly address the problem. However, the problem may not be as simple as re-coding a file, e.g. it may be that the database you originally selected for its fast handling of queries has no way to adequately separate user data and therefore requires a new database technology! So, the scale and variation of the problem to be addressed once discovered can vary from quite minor to very significant, and that's without considering the additional operational impacts while you fix the problem (such as increased monitoring).

> It's not surprising that, given this rich stack of technology, (security) issues might arise either through the complexity of the application being assembled or through delivery pressure.

Secure Systems – Why it's hard

Securing systems is not trivial. An operating system has tens of millions of lines of code, a database server or a web server can contain several millions of lines of code. The software is resident on a complex mesh of servers, network infrastructure and multiple protocols on top of which the actual application is built that delivers the services which differentiate the organisation and provides value to your customers. So it's not surprising that, given this rich stack of technology, some 'issues' might arise either through the complexity of the application being assembled or through delivery pressures.

Unfortunately, such issues can quickly become security vulnerabilities. You only have to examine a number of prolific security industry reports [1], [4] to see that there are substantial numbers of vulnerabilities reported in web applications and enterprise software annually. There are also fewer but substantially higher impact year on year increases affecting software within technologies such as mobile platforms or Industrial Controller Equipment (SCADA). Of particular interest are reports which correlate vulnerabilities from internet facing web applications across a variety of industry sectors. In these we find common serious security issues, with one study [1] citing that 53% of all systems scanned contained a vulnerability that was potentially exploitable. Whereas a separate study of different applications [5] showed that 86% of the websites contained at least one serious flaw. So, if you take into account that such scans often check only for the 'known' bugs and flaws, then the question remains how many more latent security bugs and flaws that are not yet known are within such applications?

How can it be made secure?

The journey

The challenges of system complexity, pace of delivery and technology integration are not new to many organisations and development projects. Nevertheless, a number of organisations have successfully established secure design and development lifecycles as part of a prolific corporate commitment to reduce vulnerabilities within their products [14] [15][16]. The lessons learned from these and other programmes have resulted in an increase in the techniques and guidance for secure design and development, with a corresponding improvement in secure systems [17] for those organisations that embed these techniques.

The following sections outline some of the key steps that should be used to secure systems.

A number of organisations have successfully established secure design and development lifecycles.

Security Requirements

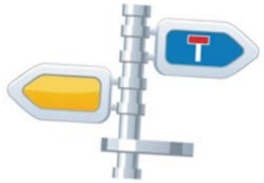
Plotting the journey

Clearly articulated security requirements are the starting point for any secure system development. These security requirements have to be related back to the actual system required by the business and its risk profile, and so are dependent upon the initial risk assessment for a system (or application), the threats, the potential vulnerabilities and the true impact to the business of the failure. This enables the subsequent secure development process to be tailored relative to the risk profile for the system, with the appropriate steps, quality and review gates.

If security requirements are going to be useful for subsequent development, then they need to be specific and not general statements such as "the application must authorise all users". It is far better to help the developers and designers with "the application must authorise users using the username, Memorable Word and PIN code". The requirement can then be further refined to constrain the subsequent functionality so that "each (web) page must check that a user's session is valid" and the "default behaviour is to deny access to all (web) pages without a valid session". Typically, this is not a quick exercise but elements can be re-used and their relationship to the business risks justifies their inclusion.

The form of the requirements need not be a pure specification document; indeed in previous projects CGI has found potential use cases or more precisely misuse/abuse cases have been helpful in communicating with system architects and developers the threats and risks to the system. The benefit is that the architects and designers clearly understand why they are implementing security controls and their relationship to the threats to the system rather than see them as annoyances that get in the way of implementing the business solution. Communication and collaboration between the business users, developers and security experts is essential to get the requirements understood and implemented as they are intended, rather than wait for disappointment at the end of the project.

With the requirements documented and an understanding of the application risk, the processes to be applied to the system development including the quality/security review gates can be defined and the next stage can be entered, which is to review the initial design.



Architectural Design

Picking the right road

Imagine this example, you are playing an online lottery game, you choose your numbers and you submit your bet then wait for the return of the results after a defined time period. You and the other customers are then presented with the winning numbers and you find out you've lost or you've won! More frequently you've lost. However, what if you noticed that the winning numbers were returned to your browser a few seconds before you were presented with the result and what if you found that if you re-submitted the bet with those returned results (very quickly after receiving the results) that you could change your numbers to the winning numbers and thereby win?

Actually this example was real but has long since been fixed and was an original flaw in the design of a gaming application. Yet it was also probably one of the simplest flaws where the designers incorrectly trusted the client (browser) with sensitive data in addition to not closing the lottery session before transmitting the numbers to the customers.

Unfortunately, such design issues are not always so simple and indeed not that easy to fix. The design issues can be caused by incompatible technologies or alternatively incorrect assumptions about the environment in which they will be working.

Take for example the hospital software that pulls together a patient's records from different hospital departments with some records on file systems and others in databases. Here the designer has to carefully define the complex security rules (e.g. based on the user's role, the doctor's and patient's permission), how the application code implements these rules and also how those 'permissions' translate across the technology tiers: the database, operating system file systems and networks. If the security is only implemented within the application then the users may just navigate around it and directly access the records on the server. If it is at too low a level, such as within the network, then you won't be able to enforce complex security policies.

In which layer do we trust?

Systems are composed of multiple software layers, often dependent upon the others for certain functions, including security. Assumptions about these layers and about how they will be configured or used can often undermine the overall system security.

One such security issue, that's frequently proven to be problematic, is the use of cryptographic libraries (such as TLS/SSL) [2]. A recent study [11] on an Android platform demonstrated the inadequate 'certificate' checking by banking applications (amongst other applications) to validate and setup a secure channel between the device and the organisation's application, leaving the users vulnerable to interception[1].

1 In reality such an attack also requires the subversion of some of the components of the network infrastructure between the user and the application (such as on a wireless network), but then the server certificates were used as a secondary protection mechanism to ensure only the authentic application is being communicated with.

> More disconcerting was that 26% of all (29.8 million) library downloads contained vulnerabilities.

A second issue is the assumption about the number and frequency of vulnerabilities that may be latent within the libraries and frameworks that you use as part of your system. For example, a recent study by an application security company [3] found that 37% of the most popular frameworks and libraries (used primarily for building web based applications) contained at least one vulnerability. Possibly more disconcerting was that 26% of all (29.8 million) library downloads contained those vulnerabilities. This problem is increasingly likely in mobile devices where libraries, such as Webkit, have been shown to contain serious flaws affecting the integrity of a number of mobile (browsers) and subsequently the devices themselves.

A third security issue is the introduction of malicious code into the system or where perhaps that application becomes malicious (i.e. once deployed). Mobile applications often have integrated (via a Software Development Kit) connection to advertising, in order to generate revenue. Unfortunately, some such benign apps have also been discovered to connect back into a malicious ad network (such as BadNews [13], [10]) that will serve up malware potentially compromising the customer's device.

Fortunately, there are a number of steps that the system designers should be going through at this stage to avoid such failures.

Architecture Checklist:

1. Independently analyse the system design against specific threat or attack patterns. Microsoft's STRIDE is one such generic pattern frequently used to stimulate that process, but it's also important to look at the specific threat patterns within your industry, e.g. for banking and retail applications the 'man in the browser attacks'.
2. Look at the technologies used and understand where there may be incompatibilities, particularly with security controls such as cryptographic libraries/APIs and access control. Alternatively, as is the case for mobile development, the design may be considering using common functionality to develop applications (HTML5/ Javascript) that may undermine the security on specific platforms. Or it may use the specific mobile platform toolkit that may initially make the design stronger, but will need any operational platform patching to be carefully considered to avoid applications on one platform being patched while there is lag in fixing the other platforms.
3. Understand and capture how the architecture is built up. How and where the important data flows and is stored? Where the security controls are placed? What assumptions are being made at the network, in the operating system, in the application (and its software stack of libraries/frameworks) and what assumptions are we making about the user and their environment.

We might suppose that our corporate environment is free from hardware or network level attacks but you only have to look at recent news reports of such attacks on retailers and banks [6] to realise that we may need to check that assumption.

4. Verify and understand the design and assurance of all the components. How have they been developed? How have they been validated or tested? And how are they being used? For example, have you tested the applications and components according to your standards? Do they have any third party assurance (and what confidence does that give?). As per the previous examples do our developers understand the risks, issues and mitigations and are they using the APIs or applications in the right way?
5. What is the provenance of the components and applications? Understand how the components to be used have been or will be built. Will you be able to understand who has had access to the code? For example, has the code been hosted on a potentially insecure public cloud with limited auditing and weak access controls to the code? Have they been developed, tested and managed in a way that we know is free from tampering?

Develop & Test

Avoiding damage along the way?

Returning back to our previous online gambling example, imagine that the flaw has been fixed and instead of winning you find yourself losing every time and the account balance dwindles away. However, what if now you were allowed to submit varying amounts of money for each bet and instead of betting positive values you entered negative numbers? You might be surprised to find that another real (but now fixed) application instead of debiting your account every time you lost, credited the balance by the same (negative) number bet, thereby increasing the account balance!

While this could arguably be considered to be a logical flaw, it demonstrates one of the most basic security mistakes made by developers – essentially "trusting the input". Indeed many of the most serious vulnerabilities discovered in systems software are caused by trusting input, be that from a user, another application or a network protocol/packet. But this is only

one potential development issue, there are also many other types associated with web applications (Cross Site Scripting (XSS), broken authentication mechanisms, forced browsing etc), with databases (SQL injection), within the code in operating systems (such a buffer overflow, race conditions).

However, unlike the earlier design flaws, these programming errors can be more readily addressed during the development process by the programmer, as long as they are provided with the right tools, process, standards and training.

Development Checklist:

1. Coding standards have been defined and the code is checked to ensure it adheres to those standards and is understandable for maintenance purposes. Automated tools can be used to check code against the defined standard (e.g. checkstyle).
2. Train the developers to write secure code, by understanding common attack patterns, such that they validate and sanitize input or use trusted security APIs from organisations such as OWASP. For instance there are increasingly good guidelines for mobile application development from both the manufacturer (e.g. Apple) and governments (CESG [12]).
3. Train and give the developers access to both security testing tools (e.g. those used to test and manipulate protocols and application input) as well source code analysis tools for potential security flaws. Some of the analysis tools can be run outside of working hours to reduce the downtime for development.
4. Ensure that the team has a security champion within it, someone who is prepared to mentor (and train) other team members, help resolve security questions and share best practice/ lessons learned.
5. Use the output of automated tools (or third party services – including those embedded within mobile app stores) such as source code analysis to look for common recurring problems that might indicate that developers don't fully understand certain issues and use this to focus the ongoing training and sharing of best practice.

Assurance

Ready to proceed?

So you've articulated your security requirements and you've translated them into a security architecture. You've then developed the system (encouraging the developer to use security testing and source code analysis tools) during the development process. The last stage is to focus on the real world security testing, where the application or system has been integrated and all of the components in the system must be tested together to check that the security controls work.

This includes creating tests based on:

a) The original security requirements e.g. does the application stop us browsing to another users bank account summary when we aren't authenticated?

b) Common security issues e.g. does the application stop us injecting commands into the application to extract from or put information into the database? Has the data been stored in the mobile device using encryption?

c) Specific applications or specially developed security controls e.g. does the system interface work with the cryptographic module/library correctly and does it handle and present the challenge/response for the pin pad correctly?

Third party software should be of particular focus as it may itself contain bugs or be incorrectly configured when used with the rest of the application stack. Unlike the code the organisation has developed, it may not have been subject to an architectural review and constant testing while in development. Additionally, it may require testing for the presence of malicious applications or software as per the earlier mobile example [13].

Particularly for mobile applications, the security testing should be supplemented with additional steps to review the open-source meta-data about the application and developer (i.e. to establish how trustworthy they are). This should be followed by a local analysis of how the data and device capabilities accessed, along with a source code and network analysis.

In most instances, an independent team of testers best perform this step. The broader the experiences of the security testing team (within and outside of your industry sector) the better the testing result.

> Developing the skills and security culture is one of the most important activities needed to develop a secure system.

Security Culture & Skills:

Setting the tone

Developing the skills and security culture is one of the most important activities needed to develop a secure system. All of the people involved in building the system must believe that a secure system is worthwhile, and not only will it protect the business and the customers, but also they must have the right skills and training to achieve it. It is also something that must be right at the inception, throughout the project and long after it is completed and embedded into the team.

So far in this chapter the focus has been on designers and developers, yet in reality, when it comes to constructing a system, we need to know that many of the roles involved (e.g. project managers, contracts, business managers) all believe in the value of the steps outlined. Otherwise they may be tempted to trade-off without understanding the implications – "early delivery rather than compete security testing" or "choose not to check third party secure development practices before purchasing for cost reasons" or "perhaps not check the identity of all users in the interests of usability".

Creating a security culture is not an easy thing to achieve, especially if you have a diverse or large geographically distributed organisation. However, CGI has found the focusing on the following steps helps:

1. Getting buy in from senior stakeholders early and clearly articulating the benefits to them.
2. A clear message from the senior stakeholders disseminated to the rest of the organisation.
3. Focused training for developers (on security vulnerabilities/common patterns and security test tools).
4. Giving frequent feedback to the developers and using it to refine the training of the existing development or next generation team so that the lessons learnt are not lost. Be careful to capture and manage this knowledge with incoming or new suppliers and teams otherwise you may find yourself starting from scratch all over again.
5. Giving frequent feedback and communicating with the rest of the 'development' team on the security progress. The benefits and successes, such as the number of bugs and flaws avoided through design and development reviews and testing.
6. Include security in reporting mechanisms. Staff and managers will always respond to what they are measured on.

Changing Landscape

Increasingly, for many organisations, the software development lifecycle has been compressed and changed from what were once monthly cascading (or waterfall) phases to now weekly updates with daily stand-ups and an iterative and rapid software development (AGILE) process.

A number of organisations have needed to adopt AGILE to increase the pace of development for the organisation and have integrated the security lifecycle steps presented earlier in the chapter. For example, while some AGILE processes are well suited to integrating security, such as the code development phase with its rapid development and iterative testing, by contrast the architecture analysis can seem at odds with the iterative nature of AGILE. The reality is that design reviews and code clean-up need to be added to the backlog to become part of scheduled sprints. Of course in order to get these activities onto the backlog, you still have to use the user stories to develop and articulate the security requirements, either by establishing them around the functionality required or alternatively as a constraint upon that functionality. Requirements particularly need to be given sufficient priority so that they don't fall below the achievement line of each sprint. Given the rapid nature, individual sprints must focus on completing the most relevant parts of the previously described steps (e.g. design review, testing, code analysis) rather than attempt to tackle all of the steps within one sprint.

However, the key to establishing any secure development is still to get user buy-in and appropriate prioritisation of these activities by the team.

Final Thoughts

Based on CGI's experiences there is no quick route to building a secure system. However, as CGI and many other organisations have found, by focusing on the key steps within this chapter, you can significantly increase the chances you will achieve a secure system and may also decrease the overall cost of development.

The security requirements are the starting point, linked to the enterprise risks and threats from which a system can be developed. These requirements need to be proportionate and relevant to the risk, e.g. a mobile application can have a very different set of requirements to that of an internal facing web application.

Eliminating significant flaws during the design or architectural review stages ensures you don't face 'insurmountable' security problems once the system is deployed or just prior to deployment. By contrast, the secure development practices and independent security testing should enable the development team to produce a quality product with minimal bugs that are also less costly to fix once a system is 'live' and with customers.

> The key ingredient is ensuring that you have established the secure development practices as being relevant and 'alive' within the organisation.

However, the key ingredient to all of this is ensuring that you have established the secure development practices as being relevant and 'alive' within the organisation:

- You have established and defined the development process so that it is repeatable, constantly improving and adapting to the business.
- You have gained and continue to receive support from the management and organisation (based on improved security results! – remember to measure).
- You have focused on your organisation's needs (the technologies used, the supply chain/partners, the agility needed) and most importantly the risk the organisation is prepared to accept.

Top 10 questions to ask of your organisation:

1	Do you follow a defined process to build secure systems? Have you defined the appropriate review/quality gates?
2	Do you have the support of senior management, Project Managers, Contracts and Developer teams for a secure development process?
3	Do your suppliers understand and support this process?
4	Do you regularly and clearly identify the realistic threats and risk to system data and functionality?
5	Have you provided clear security requirements that can be tested against?
6	Have you independently reviewed the architecture? What threats have been modelled? Do you understand how the system protection works across all technology? What assumptions have been made about the environment and ongoing operations/maintenance/customer interaction with the system?
7	How have you developed the code? Have you defined secure coding standards? Have you trained your development teams to understand the key risks and threats and how to protect against them? Have you given them security testing tools and source code analysis tools to check for problems?
8	Do you conduct independent testing that simulates real-world attacks?
9	Do you measure and feedback regularly the results and lessons learnt from testing and source code analysis into developer training?
10	Are you developing the right skills and culture to consistently build secure systems? Do you have the right security champions in place?

References

1. *2013 Internet Security Threat Report, Volume 18*; **http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v18_2012_21291018.en-us.pdf**
2. *The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software*; M. Georgiev et al; 2012; **http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf**
3. *The Unfortunate Reality of Insecure Libraries; Aspect Security*; March 2012 **http://cdn1.hubspot.com/hub/203759/docs/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf**
4. *HP 2012 Cyber Security Risk Report*; 2012; **http://www.hpenterprisesecurity.com/collateral/whitepaper/HP2012CyberRiskReport_0213.pdf**
5. **https://info.whitehatsec.com/2013-website-security-report.html**
6. *Scammers bug retail registers with $40 keylogger devices;* SC Magazine; October 2013; **http://www.scmagazine.com/scammers-bug-nordstrom-registers-with-40-devices-to-skim-card-data/article/316001/**
7. *Anonymous speaks: the inside story of the HBGary hack*; Ars Technica; February 2011; **http://arstechnica.com/tech-policy/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack/**
8. *TJX, Visa Agree to $40.9 Million Payout for Data Breach*; BankInfoSecurity; **http://www.bankinfosecurity.co.uk/tjx-visa-agree-to-409-million-payout-for-data-breach-a-648**
9. *Chinese Hackers Suspected In Long-Term Nortel Breach*; Wall Street Journal; February 2012; **http://online.wsj.com/news/articles/SB10001424052970203363504577187502201577054**
10. *Bad News for Android as Fake Ads Target Google play; Mobile World Live*; April 2013; **http://www.mobileworldlive.com/badnews-for-android-as-fake-ad-network-targets-google-play**
11. Attackers can slip malicious code into many Android apps via open Wi-Fi; ArsTechnica; Sept 2013; **http://arstechnica.com/security/2013/09/attackers-can-slip-malicious-code-into-many-android-apps-via-open-wi-fi/**
12. *End User Application Security Guidance for iOS*; CESG, November 2013; **https://www.gov.uk/government/publications/end-user-devices-security-guidance-apple-ios-application-development/end-user-devices-security-guidance-apple-ios-application-security-guidance**
13. "*Mobile Devices = New Malware and New Vectors*"; Palo Alto Networks; August 2013; **http://researchcenter.paloaltonetworks.com/2013/08/mobile-devices-new-malware-and-new-vectors/**
14. *Cisco Secure Development Lifecycle; Cisco;* Retrieved November 2013; **http://www.cisco.com/web/about/security/cspo/csdl/index.html**
15. *Oracle Software Security Assurance; Oracle;* Retrieved November 2013; **http://www.oracle.com/us/support/assurance/overview/index.html**
16. *Microsoft Security Development Lifecycle; Microsoft;* Retrieved November 2013; **https://www.microsoft.com/security/sdl/default.aspx**
17. *The Trustworthy Computing Security Development Lifecycle. The Benefits; Microsoft;* March 2005; **http://msdn.microsoft.com/en-us/library/ms995349.aspx#sdl2_topic4**

Do you measure and feedback regularly the results and lessons learnt from testing and source code analysis into developer training?

“You only have to read about the increasing number of publicly quoted data breaches to see that an insecure system is not good for business.”