October, 2009

SecurityActs

The Magazine for IT Security

made in Germany
free digital version
www.securityacts.com

© iStockphoto/bjalmeida

APPLICATION SECURITY

www.diazhilterscheid.com

How high do you value your and your customers' data? Do your applications reflect this value accordingly? Accidental or deliberate manipulation of Data is something you can be protected against.

Talk to us about securing your Systems. We will assist you to incorporate security issues in your IT development, starting with your system goals, the processes in your firm or professional training for your staff.

as@diazhilterscheid.com

© iStockphoto.com/abu

Co-Founder of ISSECO (International Secure Software Engineering Council)

Dear readers,

Security Acts is the challenge of producing a high-quality magazine for professionals in IT Security, which is made by and issued for the people involved in IT Security. This online magazine is free of charge and will finance itself through adverts.

The challenge of such an adventure is to find the right authors with the right subjects, the right readers and - last but not least - a supporting community.

Our editorial board include Prof. Dr. Sachar Paulus, Aaron Cohen, Manu Cohen and Markus Schumacher, as well as our chief advisor Stephan Goericke, who will support strongly the right development of the magazine.

We hope to get the support of the global IT Security community and ask them to subscribe to the magazine and to make the magazine's link known to their contacts.

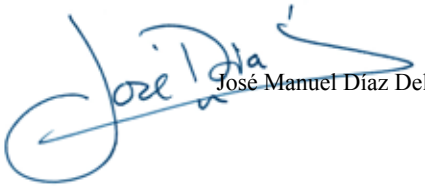
The magazine is planned to be issued 4 times a year and there will only be an online version.

We are open to all authors around the world. If you have a good subject in the field of IT Security, please don't hesitate to contact the editorial staff.

Please also contact us if you have any further questions or just to give us some hints.

I want to thank all the authors and sponsors of the magazine. Without their help we could not have produced this first issue.

Yours sincerely



José Manuel Díaz Delgado

Contents

AJAX makes applications more difficult to secure

by Manu Cohen

AJAX is the new hot technology concerning web applications. It allows the client to do much more than before and have a much better user experience.

AJAX is based on XmlHttpRequests that the browser creates while the page is being presented on the browser. The client is not aware that so many requests are being sent in the background. Ajax is a Java script technology running mostly on the client side, and on the server side the following question arises: Will the average AJAX-enabled web-application be able to tell the difference between a real and a faked XmlHttpRequest?

The answer is NO. AJAX is a client side technology and as we all know the client side should not be trusted. Any web app should behave this way, but here the request is done by a script which can be injected.

We all know how difficult it is to provide application security for traditional server applications. For AJAX it is double the effort. The main problem is that with AJAX the user in unaware of the many AJAX requests executed in the background. With a simple AJAX script injection a "virus" can be created or other unwanted behavior concerning the power of other servers on the web can be injected to the site.

AJAX is advancing rapidly and new frameworks are introduced frequently. Microsoft releases its new AJAX framework with .Net 4.0 and it promises to provide an answer for security issues. For the time being, there is no simple solution.

When using AJAX an intensive validation effort must be done, otherwise there will be no choice to make a decision between security versus user experience.

Proper design allows AJAX to be used in a sandbox. This means that less sensitive areas of the application can enjoy AJAX, but an "AJAX Firewall" is built around the sensitive business logic and information. For example, there will be no AJAX-enabled web service that exposes sensitive information.

If you need to implement some AJAX, remember that AJAX enables new XSS capabilities and therefore server validation must be much more strict. Not only the body of the http packet must be validated, but **all** of its headers. Never trust a third party. AJAX applications fetch information from various untrusted sources such as feeds, blogs, search results. If this content is never validated prior to being served to the end browser, it can lead to dangerous cross-site exploitation.

With AJAX, much of the logic shifts to the client-side.

This may expose the entire application to some serious security threats.

The urge for data integration from multiple parties and untrusted sources can increase the overall risk factor as well: XSS, XSRF, cross-domain issues and serialization on the client side, and insecure web services, XML-RPC and REST access on the server side (This is true for any WS). Conversely, Ajax can be used to build graceful applications with seamless data integration.

However, one insecure call or information stream can backfire and end up opening up an exploitable security hole.

These new technology vectors are promising and exciting for many, but even more so for attack, virus and worm writers.

To stay secure, first answer the question: Is AJAX really needed? Secondly, make sure it lives in a sandbox, and thirdly make sure your developers are paying the nessesary attention to implementation details and take security into consideration.

To summarize: Protecting an application is difficult. This is one of the reasons why so many applications are unsecured. With AJAX it is even harder. Before using AJAX ask yourself: are you ready for the security challenges?

For more information:

http://www.owasp.org/index.php/Testing_for_AJAX_Vulnerabilities_(OWASP-AJ-001)

http://www.net-security.org/article.php?id=956

http://www.darknet.org.uk/2006/04/ajax-is-your-application-secure-enough/

http://www.2bsecure.co.il/NetUG/the%20need%20for%20ajax.pdf

Manu Cohen-Yashar is an international expert in application security and distributed systems. Currently consulting to various enterprises worldwide and Germany banks, architecting SOA based secure reliable and scalable solutions.
As an experienced and acknowledged architect Mr Cohen-Yashar was hosted by Ron Jacobs in an ARCast show and spoke about interoperability issues in the WCF age. Mr Cohen-Yashar is a Microsoft Certified Trainer and trained thousands of IT professionals worldwide.
Mr Cohen-Yashar is the founder of an interoperability group in cooperation with Microsoft Israel in which distributed system experts meet and discuss interoperability issues. http://www.interopmatters.com/
A wanted speaker in international conventions, Mr Cohen-Yashar lectures on distributed system technologies, with specialization on WCF In which he is considered one of the top experts in Israel. Manu won the best presentation award in CONQUEST 2007.
Mr Cohen-Yashar is currently spending much of his time bringing application security into the development cycle of major software companies (Amdocs, Comverse, Elbit, IEI, The Israeli defense System...)
Mr Cohen-Yashar is giving consulting services on security technologies and methodologies (SDL etc)
Mr Cohen-Yashar is known as one of the top distributed system architects in Israel. As such he offers lectures and workshops for architects who want to specialize in SOA, and leads the architecture process of many distributed projects.

© iStockphoto.com/timhughes

An Overview of Software Supply Chain Integrity

by Paul Kurtz

Providing high level of assurance in ICT systems increasingly relies on ensuring consistent integrity practices are applied across the software supply chain

Commercial software underpins the information technology infrastructure that businesses, governments and critical infrastructure owners and operators rely upon for even their most vital operations. For that reason, enterprise customers are rightfully concerned about the security of commercial software and the potential for its exploitation by those seeking to maliciously disrupt, influence or take advantage of their operations.

As the software industry has become increasingly globalized, questions have been raised about what additional product security and brand risks are introduced by the increased distribution of software development activities, how these risks should be assessed, and what proactive measures can minimize their occurrence.

These questions are of interest to suppliers and customers alike and have recently been aggregated under the label of "software supply chain integrity."

However, the concept of software supply chain integrity and its key components of "software integrity" and "software supply chain" are not clearly defined, thus creating significant challenges for customers and suppliers working to identify, compare, communicate and evaluate software integrity best practices. Recognizing this gap, SAFECode (http://www.safecode.org/) has developed the first industry-driven framework for analyzing and describing the efforts of software suppliers to mitigate the risk of software being compromised during its sourcing, development or distribution.

What is Software Integrity?

Software integrity is an element of software assurance, which SAFECode defines as "confidence that software, hardware and services are free from intentional and unintentional vulnerabilities and that the software functions as intended."[1] Software assurance is most frequently discussed in the context of ensuring that code itself is more secure through the application of secure software development practices.

1 SAFECode, "Software Assurance: An Overview of Current Best Practices," February 2008.

However, eliminating software vulnerabilities through secure development practices represents only one aspect of software assurance. Another key consideration is the security of the processes used to handle software components as it moves through the software supply chain.

In practice, software assurance involves a shared responsibility among suppliers, service and/or solution providers, and customers encompassing three areas:

- **Security**: Security threats are anticipated and addressed in the software's design, development and testing. This requires a focus on both quality aspects (e.g., "free from buffer overflows") and functional requirements (e.g., "passport numbers must be encrypted in the database").
- **Authenticity**: The software is not counterfeit and customers are able to confirm that they have the real thing.
- **Integrity**: The processes for sourcing, creating and delivering software contain controls to enhance confidence that the software functions as the supplier intended.

Software integrity practices are essential to minimizing the risk of software tampering in the global supply chain.

The Challenge to Software Integrity

Governments, businesses and consumers purchase IT solutions (systems, products or services) that are a complex collection of inter-related components assembled from hardware, software, networks, cloud services and outsourced operations. Throughout an IT solution's lifecycle, which can extend over more than a decade, many individuals have legitimate access to its components and operations.

The intentional insertion of malicious code into the solution's software during its development or maintenance is often referred to as a supply chain attack. A supply chain attack can be directed at any category of software, including custom software, software delivering a cloud service, a software product, or software

embedded in a hardware device.

Software is packaged as a collection of files. To be successful, a software supply chain attack must result in either: a) the modification of an existing file(s); or, b) the insertion of an additional file(s) into the collection of software files.

Reports[2] that have considered supply chain attacks have concluded that: 1) there is no one way to defend against all the potential attack vectors a motivated attacker may identify; 2) focusing on the place where software is developed is less useful for improving security than focusing on the process by which software is produced and tested; and 3) there are circumstances when the insertion of malicious code would be almost impossible to detect.

It is important to recognize that while there is a risk that someone with malicious intent could attack software during its development, experts[3] have concluded that supply chain attacks are not the most likely attack vector. For example, the practice of hackers or other malicious actors finding and exploiting existing vulnerabilities remains the most common method of attack.

Software Integrity Control Points

Sophisticated IT solutions have much in common with other engineering undertakings. Each IT solution is a collection of components. Each component or its parts can be: a) developed by its supplier or on that supplier's behalf by their subcontractors; or b) licensed to the supplier by another vendor or obtained from Open Source repositories; or c) acquired outright by the supplier.

Yet, this complexity can be organized. IT suppliers have natural control points within software supply chains. To identify these, consider that each software supplier controls three links of the supply chain. For these three links each supplier takes similar actions:

1. **Supplier Sourcing**: Select their sub-suppliers, establish the specification for a sub-supplier's deliverables, and receive software/hardware deliverables from sub- suppliers;
2. **Product Development and Testing**: Build, assemble, integrate and test components and finalize for delivery; and,
3. **Product Delivery**: Deliver and maintain their product components to their customer.

As such, suppliers have an opportunity to apply integrity controls at each of these key links in the supply chain. For instance, a supplier can conduct acceptance tests on components received from their suppliers, and release tests on the components they deliver to their customer.

To be effective in today's complex global supply chains, software integrity processes and controls must be designed to be independent of geography, accommodate diverse sources of software components, and extend from a vendor's suppliers to its customers.

An Industry Imperative

Suppliers are aware of threats to their products and are, consequently, extremely protective of their code base – not only is the integrity of their products at stake but also their highly valuable intellectual property and brand. As such, suppliers delivering software have significant experience implementing powerful management, policy and technical controls that reduce the risk that their code can be compromised.

Yet, while individual software companies have integrity assurance programs in place, there has been little industry-led effort to identify and share best practices for implementing integrity controls or to provide customers with more clarity into how the industry is addressing this issue.

This is a critical gap that SAFECode is currently addressing with a focused effort to identify the threats, assess the risks, share current practices for mitigating those risks, and develop process guidelines that other software companies should consider adopting to protect the integrity of the software they produce through the global supply chain.

Given that the complexities and interdependencies of the IT ecosystem require software suppliers to not only be able to demonstrate the security of products they produce, but also evaluate the integrity of products they acquire and use, every software supplier has a significant stake in the identification, communication and evaluation of best practices for ensuring software integrity. The challenge is to create practical but effective methods that build on a broad understanding of the dependencies and threats along the complex software supply chain. Ultimately the industry-wide adoption of well-defined software integrity practices should lead to increased customer confidence in the security of IT solutions.

2 "Mission Impact of Foreign Influence on DoD Software," U.S. Defense Science Board, September 2007. "Foreign Influence on Software: Risk and Recourse," Center for Strategic and International Studies, March 2007. "Framework for Lifecycle Risk Mitigation For National Security Systems in the Era of Globalization," U.S. Committee on National Security Systems, November 2006.

3 "Mission Impact of Foreign Influence on DoD Software," U.S. Defense Science Board, September 2007. "Foreign Influence on Software: Risk and Recourse," Center for Strategic and International Studies, March 2007.

Paul B. Kurtz is executive director of the Software Assurance Forum for Excellence in Code (SAFECode) and a partner at Good Harbor Consulting. A recognized cyber security and homeland security expert, Kurtz served in senior positions on the White House's National Security and Homeland Security Councils under Presidents Clinton and Bush and as an on-air consultant to CBS News.

Prior to joining Good Harbor, Kurtz served as the founding Executive Director of the Cyber Security Industry Alliance (CSIA), an international public policy advocacy group dedicated to ensuring the privacy, reliability and integrity of information systems. Previously, Kurtz held a number of positions in the White House, most recently serving as special assistant to the President and senior director for critical infrastructure protection on the White House's Homeland Security Council (HSC).

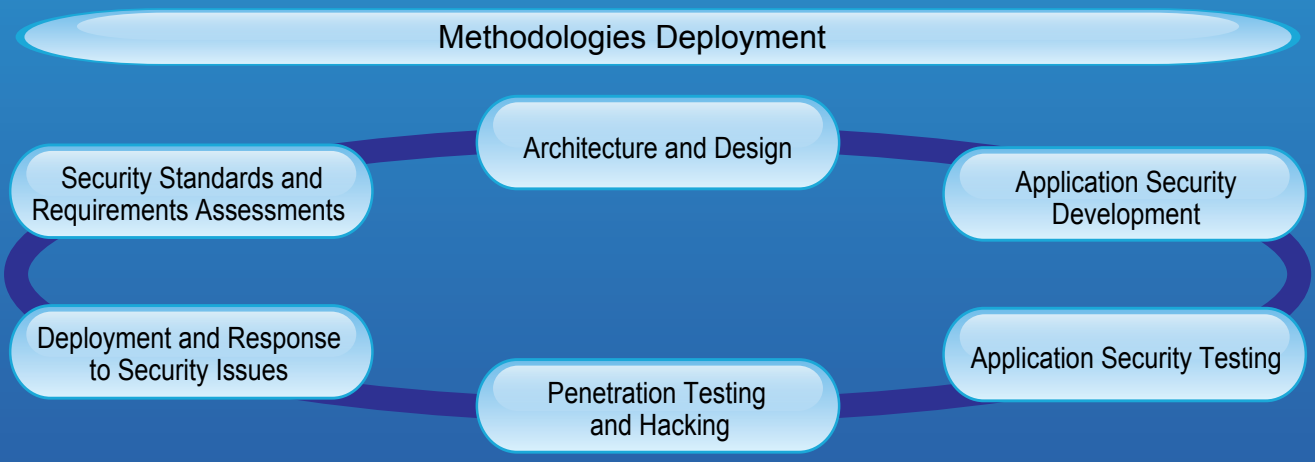
We See the Whole Picture...

Addressing Application Security from a Holistic Perspective

Applications today can no longer rely on the infrastructure to secure their assets. The applications must be built with integrated security design and that is a great challenge.

We can help you to secure your software!

SELA, a founding organization of ISSECO®, offers you comprehensive services helping you to integrate application security into the product life cycle. The services are provided for each and every step of the development life cycle:



Available Courses:

- ISSECO® Certified Professional for Secure Software Engineering
- Application Security Testing
- Application Security Design
- Introduction to Information Security
- Application Security using C++
- Web Services Security
- Windows 7 Security
- Application Security using the Microsoft .NET Framework
- Vista Security

Next *ISSECO® Certified Professional for Secure Software Engineering* Courses

Country	City	Date
Israel	Tel Aviv	November 17-19/2009
Canada	Toronto	November 25-27/2009
Israel	Tel Aviv	December 20-22/2009
India	Pune	January 20-22/2010
Singapore	Singapore	January 25-27/2010
Israel	Tel Aviv	February 7-9/2010
Canada	Toronto	February 24-26/2010

solutions@sela.co.il
www.sela.co.il/en

ISSECO® SECURE SOFTWARE ENGINEERING

SELA TECHNOLOGY CENTER

Microsoft GOLD CERTIFIED Partner
Learning Solutions
Networking Infrastructure Solutions
Information Worker Solutions
Custom Development Solutions

The Science of Secure Software

by Prof. Dr. Sachar Paulus

Lately, a number of scientist colleagues and myself were preparing a proposal for a project that will hopefully be funded by the EU commission. The project proposal was about measuring security.

„Now how can you do that?“ some of you would probably claim, thinking „This is simply impossible“. Others might say „Well, nothing easier than that, we have been doing this for years“. Actually, I think both are wrong. What we measure today is mostly insecurity when security went wrong. This applies for virus outbreaks, thefts, fraud, and so on. We rarely measure the positive state of security, or more precisely, the negative state of no security breaches. But there are ways of doing this.

One very interesting approach in software security is the use software quality metrics that can be used to deduce a potential security level of the software. For example, the number of input channels is one, another is the number of output channels, and of course the number of validated input / output channels. This is the way we need to think in order to improve security.

However, developing secure software is still considered by most people to be an art, and not a science. This is the flip side of the great community thinking in the security arena: people that think they make the difference. And indeed, in many situations this is exactly the case. But this means we will never ever reach a ubiquitous state of security in IT. For one simple reason: individual knowledge / competencies in people‘s minds don‘t scale!

To reach a society-wide status of security in information technology, we need to standardize. We need to standardize security measures, so that people can better understand, we need to standardize security development to avoid really stupid mistakes, we need to standardize security education to scale, we need to standardize the measurement of security in order to become transparent and comparable in what we do regarding security. Remember: we want to scale.

Only when we scale and reach out for standardization, will we be able to „reproduce“ security - which is a major requirement for becoming a science.

Don‘t get me wrong: I do admire those guys who are able to hack a pretty well-secured web server in minutes, and we need them to „put the finger in the wound“ as we say in German. But they need to be the tip of the iceberg, and there must be an army of security soldiers that are able to address the standard stuff. Scale.

Another important requirement for a „science“ is measurability. Besides standardization, measurability helps to get understood by the masses. Just think about the CO2 emission per km metric to measure the environmental friendliness of a car. What about a „privacy data emission per transaction“ metric? Sounds weird? Well, we will need to accommodate. This is the type of thinking that will bring us forward.

So: yes, I believe we can measure security. It may not as simple as in the previous example, but that is the way to go.

Sachar Paulus is Professor for Corporate Security and Risk Management in the department for Business Administration at Brandenburg University of Applied Sciences. Sachar Paulus has a Ph.D. in number theory and several publications on cryptography. He was more than 13 years in business, 8 of which with SAP, the world’s largest business software manufacturer, in various positions related to security, among others Senior Vice President Product Security and Chief Security Officer. He was member of the RISEPTIS advisory board of the EC, member of ENISA’s permanent stakeholder group and is one of the authors of the Draft Report of the Task Force on Interdisciplinary Research Activities applicable to the Future Internet of the EC. He is also President of ISSECO, a non-for-profit organization aiming at driving secure software development, and standardizing qualification around secure software engineering.


© iStockphoto.com/caracterdesign

Practical Application Security

by Manu Cohen

Application Security is a problematic subject.

It is a non-functional requirement, so it cannot be presented to a customer, and it is expensive. The management feels that money is being spent without tangible results, and the developers feel that security is a pain so they will do anything to avoid it.

When being asked "what is application security?", so many different answers are given …

So how can application security be implemented? In this article I will show how the broad concept of application security can be translated into simple tangible tasks.

Application security requires design. If it is considered at the early stages of the product life cycle, it can be implemented without significant costs. If only realistic threats are mitigated, the security expenditure can be reasonable.

Still the questions "what is application security?" and "what exactly should be implemented?" remain open.

To facilitate the practical implementation, application security was divided into 10 major chapters.

Input validation

Most attacks we know today and many future attacks begin with malicious input.

In this category we find all the injection attacks, buffer overflow attacks, some DOS attacks and many more.

Simple input validation would prevent the attack. Simple measures can prevent huge threats.

There are two types of validation - white box and black box. White box means that there is a known pattern to the input. Anything outside of the pattern is considered malicious. This is a very effective method of validation, as the pattern can be easily checked. Black box is less effective but more frequently used. In this case there is no pattern. All that exists is a known list of attacks. If the input looks like a known attack, it is rejected and everything else is allowed. It is obvious why this method is less effective; it is impossible to filter out tomorrow's attacks and even known attacks are very difficult to identify.

Whenever possible, data should be designed to have a pattern, as validation is much stronger. In client-server applications, validation can be done on the server and on the client. Client-side validation should never be considered as a security measure. Its only purpose is to help the user. An attacker can easily bypass client-side validation.

Security validation must be done on the server and it must be carried out as early as possible. There are many security tools like dynamic analyzers that actually check for non-validated data flows in the system.

The rule is simple: Whenever data is received from an untrusted source (i.e. user), it should be validated.

Authentication

Authentication means identification of entities. Entities can be users, devices or applications. There are different authentication technologies for the different types of entities.

Usually users have only user name and passwords, which are the weakest type of credentials. In a domain environment users are authenticated against an Active Directory. Outside the domain an identity management database should be established. Sometimes users have stronger credentials, like a smartcard or biometrics. Governments often promise to distribute strong ID to their citizens, but until that promise is fulfilled, user name and passwords will be used for the majority of people.

Application and devices use X509 certificates created by a proper Certificate Authority (CA). An enterprise can create its own CA and issue certificates to its servers and applications and devices. Certificates depend on the trust in the CA. The idea is simple: I trust the CA. You give me a token produced by the CA that says that you are who you are. Because I trust the CA, I assume that your identity is real. Certificates also include a public key that can help transfer keys or validate a digital signature.

An application can use a third party to authenticate on its behalf. The application must of course trust the authentication provider, who can authenticate users for many applications. The user will get a single sign-on experience, and the application will be released from the burden of managing identities.

BE SAFE!

START SECURE SOFTWARE ENGINEERING

Secure software engineering has become an increasingly important part of software quality, particularly due to the development of the Internet. While IT security measures can offer basic protection for the main areas of your IT systems, secure software is also critical for establishing a completely secure business environment.

Become ISSECO Certified Professional for Secure Software Engineering to produce secure software throughout the entire development cycle. The qualification and certification standard includes

- requirements engineering
- trust & threat modelling
- secure design
- secure coding
- secure testing
- secure deployment
- security response
- security metrics
- code and resource protection.

PLEASE CONTACT:
Malte.Ullmann@isqi.org

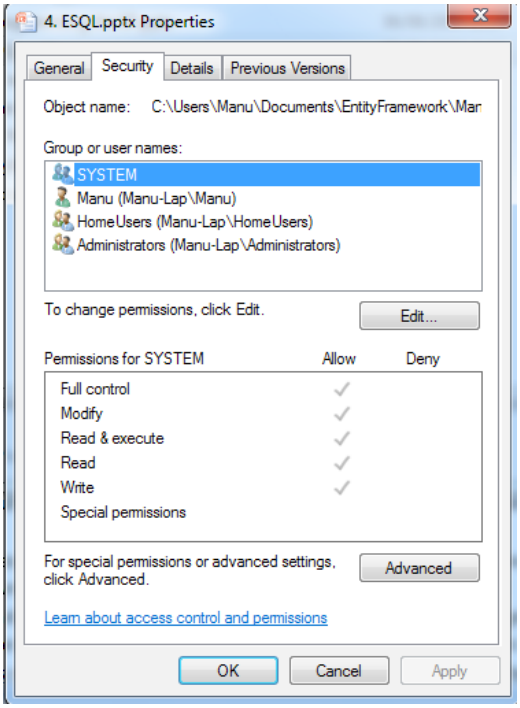
WWW.ISSECO.ORG

ISSECO®
SECURE SOFTWARE ENGINEERING

iSQI
International Software Quality Institute

Foto: ©schiffner-photocase.com

Figure – 1 Access Control List



Authorization

Authorization is the process of setting and enforcing entity permissions. There are many authorization technologies. Operating systems create a token for the user after passing authentication. This token includes a list of SIDs (Security Identifier) for the identities and the groups the user belongs to. Every resource managed by the operating system has an Access Control List (figure1).

When a user tries to access such a resource, its SIDs are compared against the Access Control List to determine the access policy.

On a client-server scenario, applications depend on a token created by the application itself or a third-party authenticator. This token includes the list of roles (groups) the entity belongs to. The application will use this data declaratively or imperatively to determine the access policy.

Another interesting authorization technology is claim-based authorization. Here the token includes a list of claims, which are actual information about the entity identity and the resources it is allowed to consume. To access a resource, an entity has to present a list of claims.The token can include a long list of claims, so the application can receive interesting information about the user from the token.

Configuration Management

The configuration holds sensitive information about the application. Connection strings that include credentials to the database are just an example.

Application use configuration files to store business policy (rule-based systems), provider types, service address and configuration, throttling info and more.

Altering the configuration can change the behavior of the application or break it. Stealing configuration can create a serious security breach. The fact that the configuration files are put on the production servers does not mean they are secure. Administrators can be manipulated and other infrastructure security problems can open access to these sensitive files.

Configuration must be treated as highly sensitive information, and thus it should be encrypted and the access to it must be logged.

Sensitive information:

What is sensitive information? To answer that question, the application and the business must be examined. The typical developer does not have full information about the business, so he cannot answer the question alone. On the other hand, the business people do not have full information about the technical aspects of the application, so they too cannot answer the question by themselves. Only deep consolidation between the two can create a list of sensitive information. Now a sensitive information policy can be enforced knowing that nothing has been left out.

Cryptography

One of the methods to protect data is cryptography. Today cryptography research is well advanced and computers have strong algorithms, but the hackers also have great tools and computing power.

Choosing the wrong algorithm can have disastrous results. To create new algorithms that cannot be broken by modern tools requires good mathematical knowledge and cryptography expertise, which are not usually available to the average developer.

Sensitive information must be encrypted using well known and tested algorithms which have been approved by distinguished institutions like the NSA.

The strength of cryptographic algorithms is a dynamic matter. What was impossible in terms of computing power a few years ago is now available in every household. The application should be able to upgrade the cryptography algorithm when necessary.

When speaking about encryption, the first thing to think about is key management.

Data is encrypted with symmetric algorithms. Asymmetric algorithms are used for key exchange and digital signature, but not for large data encryption. The one key that is used for encryption must be well protected. Protection using encryption does not solve the problem, as you still need to protect the key. This must be done using well known key management infrastructures like DPAPI or TPM.

When speaking about integrity (approval that the data source is the one expected and the data was not changed on the way), digital signature is the technology to use. Digital signature is based on the fact that the public key of the source is used to check the signature at the destination. The destination must be sure that the public key it uses belongs to the source. One of the technologies to assure that are X.509 certificates. The certificate produced by a trusted certificate authority includes the public key. The trust in the CA supplies the assurance of the key's identity.

Parameter manipulations

Distributed applications send parameters between modules. If the parameters are changed by a middle man, the application can be hacked. One great example that demonstrates this relates to a Ferrari which was bought at a price of one dollar. The trick was that the application uses the price written in the web form to actually create the debit. A hacker used an http proxy and changed this http web form field to the value of one dollar and thereby accomplished his mission.

Sometimes web sites use an index to present data, and some of these can be sensitive. When the index is put as a URL parameter, changing that parameter might reveal sensitive data.

Web forms are a great example how parameters can be changed in the URL, cookies etc and surprising effects occur.

Sensitive parameters must be protected. One way of doing so is using digital signatures. Proper design would prevent sensitive actions from being executed as a result of receiving unprotected parameters.

Exception management

Some technologies use exceptions as the root for error handling. Exceptions are great for developers and administrators, because they hold lots of information about the application and the problem that caused the exception. This advantage for the developer might be a huge problem for the user. A user may not understand the information included in the exception. It may be totally irrelevant for him. The hacker on the other hand will find this information extremely practical to design the next attack. If exceptions are exposed to the user, they are available to the hacker. Building an effective attack requires a lot of internal information about the application. Exceptions are a great source for this info.

The hacker would make the application fail and throw exceptions from which he will learn about application internals. The hacker can learn the schema of the database, different module names, important file names etc. All these will help him to design the next attack.

Exceptions should be sanitized. The user should get a general error message with no internal technical information about the application. The exception itself should be logged, so administrators can read the log and understand what went wrong and maybe fix the problem.

Logging and auditing

When the application is being attacked measures must be taken. The attack should be stopped, damage should be fixed and maybe the attacker can be caught.

The application and its administrators must understand that they are under attack.To do that, the application must distinguish between normality and abnormality. The normal activities of the application should be logged and their normal pattern should be understood. When the events logged do not match the pattern, we may be experiencing an attack. For example, in a normal scenario we know that on average 5 users per minute register to the application. If the logging shows registrations of 1000 users per minute, it should look suspicious. Maybe we are under a Denial Of Service attack.

To handle attack access to situations, sensitive information must be logged, and the logs must be protected. For example every access to sensitive database tables should be logged. It makes sense that hackers will try to access these resources. If the access is logged, it can help to find the hacker and the security breach he used.

This is why hackers will try to attack the logs themselves and try to destroy them.

Summary

Application security is more than authentication and SSL - there are a lot of issues to be taken into consideration.

When designing an application use this list of topics to create a security plan and make sure all of the above are properly handled in your application.

Choose the proper technology for each of the above issues.

Make this list available and understood by developers and testers to make sure it will be implemented and tested consistently throughout the application.

When security, which is an abstract term, becomes a list of tangible tasks, it becomes clear and practical.

See Page 5.

© iStockphoto.com/alxpin

International Secure Software Engineering Council (ISSECO)

by Petra Barzin

Certified Professional for Secure Software Engineering

Secure Software Engineering

Security concerns at the application level are a growing risk to the IT community and one of the biggest challenges for IT security in the next years. Security vulnerabilities are not limited to a few products, but affect almost all vendors and products available on the market. Although the vendors provide security patches free of charge, the roll-out of patches produces extra costs and bears the risk of new security vulnerabilities and critical incompatibilities in complex IT environments. The everlasting race against time to find security vulnerabilities before an attacker will find them or before published exploits can cause any harm before security patches are available does not seem to be the best approach to gaining confidence in the security of software. In order to win the race, the real causes of security vulnerabilities rather than their effects must be eliminated.

Security vulnerabilities may be exploited in order to steal critical company data, to distribute viruses and worms or to "rat" computer systems. Firewalls or Intrusion Detection Systems are no longer sufficient to avert danger, because they cannot prevent attacks on the applications themselves. I.e. a firewall cannot decide whether an input parameter is valid or implies a "code injection" attack. This decision can only be made by the particular application itself. Consequently, possible attacks must already be eliminated during the development of the application. Unfortunately, security aspects in the software development life cycle usually do not receive enough attention at the universities or later in the day-to-day business of software engineering in order to counter security vulnerabilities at an early stage when they are emerging.

Secure software development demands security-conscious and well-educated software architects and developers. Today, this type of qualification that attests the very best skills to produce secure software is missing. ISSECO aims at filling this gap of qualification by providing an international personnel certification standard for secure software engineering.

ISSECO Education and Certification

The scope of ISSECO is the education of people involved in the software development life cycle. The education covers all topics relevant in the area of software development. Excluded are safety matters such as perimeter and infrastructure security. Furthermore, information security management as well as cryptography are out of scope for the ISSECO foundation level, but might be addressed by further advanced level certifications (cf. section 4 "Future prospects"). Also, ISSECO does not perform security assessments of processes or IT products.

This personnel certification addresses everyone who is directly involved in the software development life cycle, i.e. requirements engineer, software architect, designer, developer, software quality manager, software tester, project manager and all related software development stakeholders.

No formal entry qualifications - such as work experience or university degree - are required to take an ISSECO training course and the examination. However, some knowledge of information technology and basics in quality assurance are expected from a candidate for Certified Professional for Secure Software Engineering.

Current accredited ISSECO training providers who offer training seminars for the Certified Professional for Secure Software Engineering include Diaz&Hilterscheid, Fraunhofer Institute IESE, Secorvo, Secunet, SQS, and VirtualForge. Other training providers that choose to support the Certified Professional for Secure Software Engineering in the future must be accredited by the ISSECO board.

ISSECO training providers and ISSECO examination providers must be independent from each other. All candidates for Certified Professional for Secure Software Engineering must take their exams at the International Software Quality Institute (iSQI).

ISSECO Syllabus

Software security is not a test case before deploying an application, and it is not an add-on feature of software. Software security is an integral component of every phase in the whole software development life cycle. Thus, the structure of the ISSECO syllabus is based on the different phases of the software development life cycle.

At the beginning the view of the attacker and of the customer need to be understood in order to be able to create secure software. In order to see with the eyes of the enemy, the Certified Professional for Secure Software Engineering must know the motivations of hackers, their skill level and resource situation, as well as typical hacker thinking when attacking systems. Furthermore, the Certified Professional for Secure Software Engineering must have understood what customers expect in terms of software security and why, in order to classify the customers' requirements. Describing use cases of the customer, his assets, threats and risks helps to avoid security conflicts which may arise when a customer has a different use case in mind than the software architects and developers.

Next, the Certified Professional for Secure Software Engineering must have a basic understanding of the different trust and threat models. Understanding the assets and its threats is a key element of threat modeling. Since threat models help to define the security objectives of an application, the Certified Professional for Secure Software Engineering must be familiar with the different threat models. In contrast to threat models, there are various access control models that describe how to constrain the ability of a subject to access or generally perform some sort of operation on an object. It's important to have these trust models in mind when designing an application as Certified Professional for Secure Software Engineering.

Furthermore, the Certified Professional for Secure Software Engineering must feel comfortable with the methodologies for secure software development. These methodologies describe the processes and practices associated with producing secure software. Processes that consistently produce secure software do not require any particular design, development, testing, or other methods. They can be applied to any development methodology or life cycle model.

The Certified Professional for Secure Software Engineering must understand the impact of security on all phases of the software development life cycle, i.e. security requirements engineering, secure design, secure coding, security testing, and secure deployment. Security must be incorporated already at the very beginning of the software development life cycle. In the requirements engineering phase the Certified Professional for Secure Software Engineering must focus on developing security requirements for the respective application. There are lots of different areas where requirements originate, and many of them are relevant to security.

Since architectural and design-level errors made in the design phase are the hardest vulnerabilities to fix and in most cases difficult to defend, the security principles and security design patterns must be well understood by the Certified Professional for Secure Software Engineering. Security architecture and design reviews help to identify problems in the application design and to discover possible vulnerabilities. Thus, at design reviews the Certified Professional for Secure Software Engineering must be able to focus on the areas of the application that have the most impact on security.

Secure coding implies that the Certified Professional for Secure Software Engineering understands which programming errors lead to vulnerabilities like Cross-Site Scripting (XSS) or injection flaws. All vulnerabilities are introduced by so-called vulnerability patterns, e.g. buffer overflow, race conditions or improper error handling. The Certified Professional for Secure Software Engineering must be able to identify, avoid and remediate them.

During security testing the Certified Professional for Secure Software Engineering verifies whether all security requirements are met and all mitigation techniques are effective. Therefore the test techniques of security testing and the correct interpretation of security testing results must be understood .

Even if security issues were considered at the initial stages of the software development and secure design and coding practice were applied during development, the security implications of deployment are often overlooked. Vulnerabilities may still arise during this final phase. Thus, a secure deployment is another concern of the Certified Professional for Secure Software Engineering as it is important for the success of the whole software development life cycle.

Once the software has been deployed, the Certified Professional for Secure Software Engineering is concerned with the implementation of a security response process in order to make sure that security issues in software installations are fixed and communicated responsibly.

Security metrics aim to quantify the security of an application. Security is a horizontal topic that involves every stakeholder, has an impact on many features, and has to be considered by the Certified Professional for Secure Software Engineering throughout the complete software development life cycle.

Last but not least, code and resource protection is a security concern of the Certified Professional for Secure Software Engineering, in order to assure the quality of software and protect it from external sabotage.

Future prospects

Besides the ISSECO foundation level certification there will be additional advanced levels, which will be defined at a later date. These advanced levels may address programming language specific security matters, IT security management or other topics. There might also be a security auditor training for assessing software development with respect to security.

Petra Barzin, Diplom-Informatikerin (computer scientist) graduated in computer science at the Darmstadt University of Applied Sciences in 1995. From 1995 until 1999 she worked at GMD (aka FhG) in the research area of Security and Smartcard Technologies. In 1999, she changed to a leading German vendor for Public Key Infrastructures (PKI) solutions where she was head of the security consulting team for four years. Afterwards she switched into product management and was responsible for the development of some selected security products.
Since October 2004 she has been working as a Security Consultant at Secorvo Security Consulting GmbH.
Petra Barzin has many years of experience in the fields of Public Key Infrastructures, secure e-mail solutions, secure Internet e-commerce protocols, Single-Sign-On, electronic signatures and compliance to the German Digital Signature Act. Petra Barzin is a certified ISC2 Information Systems Security Professional (CISSP).

© iStockphoto.com/RichVintage

The Liability of Software Producers and Testers

by Julia Hilterscheid

A recent decision taken by the German Federal Court of Justice regarding the liability of a freelancer working for a company indirectly effectively reverses the principles relating to the liability of software producers and testers, which had been applicable so far. If certain services or insufficiently tested products cause damage to the customer and require that the customer's employees have to rectify this during their regular working time, the customer can now hold the causer(s) of the damage liable with greater chance of success.

What does this mean for customers, software companies and for testers?

There are various reasons why software products end up being deficient. Firstly, it can and does happen that the requirement definitions are inadequate. When specifying the requirements for a project, the prime concern is the careful planning of the development process and a precise specification of the properties of the product in the contract. Clearly, many customers and contractors are aware of this important criterion. It is, however, all the more amazing that projects are frequently started in a "vacuum", i.e. that contracts or project documentation are only available in a rudimentary form. This means that the requirements for the program were not specified in a way that makes them capable of proof, and it will afterwards be difficult or downright impossible for either party to determine whether the delivered product is in accordance with the objectives, i.e. whether or not it is free from defects.

Another reason why software is often ends up being faulty is that the detailed concept has not been performed properly.

On the basis of a schedule, which is inevitably drafted as a rough concept at the start of the project, and in which the contents and the principal project milestones are specified, the detailed concept is created, in which the various functions are defined. Whilst the rough concept is drafted in coordination between customer and contractor, the technical department, the responsible marketing personnel and the designers, the fine concept serves the technical department and the programmers as a guideline for designing the individual steps. This includes the description of the data architecture and the business logic of the system. Based on the detailed design, the programmers will develop the procedures and functions as well as the database structures required. It is therefore possible for all project stakeholders to look at the detailed design and inform themselves at any time about contents and objectives of the project and to align their activities accordingly.

In order to develop software that is in accordance with the customer's requirements, a well functioning quality management is indispensible. Defects introduced into the software during the definition phase and programming cannot be discovered if the test process is not managed by a professional test manager.

In the development of software, the time pressure also plays an important role as a possible cause for bugs. Each tester knows the panic when time is running out at the end of projects, when the development or release of software is delayed, and if a contractually agreed delivery deadline cannot be met. The customer already threatens with a claim for damages, or – even worse – has even been able to push through a penalty clause that automatically applies when the deadline is exceeded.

As a result, management, technical departments and sales department put pressure on the tester, especially in cases where the delivery deadline cannot be put back e.g. due to legal requirements. Very often, the tester is not even responsible for the delays that have occurred; these could have been caused by bad project planning, late software requirements specified by the customer, or inadequately defined requirements. Nevertheless, the tester is the person that has to somehow cope with the virtually impossible task of delivering a product of at least satisfactory quality. Quality - due to the fact that everybody knows that quality cannot be added by the testers. He will brave the gap and hope that the defect will not occur at the customer as often as it has during testing, and he will have to release the software – albeit reluctantly. For the customer this situation can result in severe implications resulting from the defect.

Up to now, „only" the customers of the software producer had to bear the brunt in the wake of inadequately performed or omitted testing, e.g. if the program did not implement the requirements of the business processes. Development deficiencies of this type not only lead to increased customer costs, but quite frequently also to situations where the customer's employees have to try and rectify the software deficiency during operation. This had the effect that the employees were often less efficient and motivated. In addition to this, an inadequate performance on part of the software producer has an impact both on the customer's and the software producer's external presentation, since both parties can end up with a bad reputation. Moreover, it was the aggrieved customer

who was usually the one paying the bill, since he was the one who had to prove judicially or extrajudicially, which work was left undone as a result of the rectification of the damage and the financial loss that has been caused by this.

Up to now, customers have only rarely been in a position where they could assert a claim for such financially difficult to quantify damages – or if at all then not for the full amount of the damage. The time and effort expended on the rectification of faults in operation, which have already occurred or are expected to occur, was difficult to prove. This was due to the fact that the customer's employees were in the middle of the rectification process before they realized that the work to be performed and the resulting costs to be incurred were threatening to become a bottomless pit. In addition to this, employees usually still do not log the activities they had to perform at what time and for how long, in order to rectify the defects. In court, however, this sort of evidence was a prerequisite for being able to prove the circumstances and therefore for winning the case. Another problem, which in the past was certainly even more decisive in these cases, was that the customer's employees were drawing a salary anyway, which meant that the costs incurred through correcting the defects were not accounted for separately. This means that the damage was not one that could be proven in court.

Due to the uncertain outcome of court actions and because of the time and money involved, customers up to now often refrained from trying to assert their claim, especially since – as we all know – "software is always bound to be faulty". Therefore, software producers have traditionally had an advantage over their customers when the payment of damages was negotiated in court settlements.

This has now fundamentally changed due to the decision taken by the German Federal Court of Justice.

In order to successfully assert claims for damages, it is meanwhile sufficient to present the effort of the employees required for rectification of the damage and the expected effort required due to future malfunctions in operation. According to the Federal Court of Justice, the software producer, as source of the damage, must not gain an advantage from the fact that the customer's employees are in any case employed and paid by the customer.

For software producers, this will mean that in future more accurate planning of the projects will be necessary if the projects are not to become economically unviable through successful claims for damages asserted by customers. The deliverables of the software producer must therefore already be specified as precisely as possible in the contract, e.g. through a clear requirements specification. This is also of advantage for the customer, because the accurate descriptions of the deliverables also makes it possible for the customer to prove – if need be - that the software producer has not fulfilled the contractual obligations.

Furthermore, all necessary software tests must be planned at an early stage and must be performed with due care. A professionally organized quality assurance process tries to achieve the highest possible test coverage at an acceptable risk with the lowest possible number of test cases, in order to deliver the best possible software to the customer within a limited time period.

Provided that all these aspects have been taken into account, it is likely that any attempts by enterprises to claim damages against the software producers will be destined to fail, even after the new decision of the Federal Court of Justice.

Ms. Hilterscheid has been solicitor with practising licence since 1997. After stays abroad and studies in law in Berlin, she founded first the solicitor's office Hilterscheid, and one year later, together with her husband, the consultancy Díaz & Hilterscheid Unternehmensberatung GmbH. In addition, Ms. Hilterscheid has been teaching media law and copyright at the Berlin University of Applied Science.
Ms. Hilterscheid has been supporting enterprises for more than 10 years in the discretionary formulation of contracts, general terms and conditions and license agreements, and accompanies projects legally. She ensures for her clients that legal requirements are adhered to in their correspondence, on-line appearance as well as in their advertising and marketing activities.
Ms. Hilterscheid also offers seminars on the subjects of IT-law, trademark law, copyright and labor law, which can take place in-house if so desired.

www.kanzlei-hilterscheid.de

The Human Face of Security - #1

by Mike Murray

"It's The People, Stupid"

Information security is an interesting field to work in. At some point in its history, the term "information security" came to be synonymous with "computer security", and the large majority of the field became staffed with computer geeks.

This has led our industry to become incredibly technology-centric, to its ultimate detriment. Because we entered a new era of computer security somewhere around 2005, this era places the human as the primary target point of the "information security" threat landscape.

I say this often when I speak at various conferences, and I am often met with some amount of derision at the idea that people are the primary issue facing information security professionals in the coming years.

But the data backs it up. I recently spoke at an industry meeting with Kevin Haley (Symantec's Director of Security Response), who was detailing the findings in the recent Symantec Internet Threat Report[1]. The threat report spends over 100 pages talking about the newest threats against computer security and how computer systems are getting compromised.

I can sum up the data in a single sentence: in almost all cases, the compromise is happening because a user is taking some action (e.g. clicking on a link, visiting a malicious website, opening an email attachment, etc.)

This may seem obvious to most, but this has not always been the case. Only a few short years ago (way back in 1999-2003) the main threats to the internet landscape didn't involve users at all. The attacks were directly against computers and came with names like Slammer, Sasser, Blaster and Nimda. Information security professionals spent time understanding how to deploy technologies to protect their computers from direct attack – firewalls, IDS/IPS, and patch management were the important things to know if we were going to manage the risk to our business' information environment.

Many security professionals are still looking for the technological solutions that will protect them. And they are falling woefully behind the attackers in understanding the major threat against their organization.

1 http://www.symantec.com/business/theme.jsp?themeid=threatreport

For the first time, attackers are investing resources in defeating the weakness in the end user rather than the computer system. Unconfirmed reports have suggested that the writers behind some recent attacks consulted with psychology graduate students in the creation of the delivery emails and websites behind their malicious code.

Most "information security" professionals are not equipped to deal with this threat. We are, in military parlance, stuck "fighting the last war". And, to paraphrase General Eric Shinseki, if we don't like change, we're going to like irrelevance even less.

So, what do we do?

The answer to that question is going to be the ongoing focus of this column. The short answer is, of course, work with our users more and our computers less. Focus on shaping our culture and our business processes to value digital information in the same ways that we have valued physical secrets over the years.

As a simple thought-experiment: if you walked in to your average bank and approached your average (security-unaware) teller, and asked him/her for all the cash in the drawer, the response would be well-trained, well-practiced, and would (in almost all cases) lead to the protection of the money and your arrest.

If you called that same teller on the phone and asked for his/her login and password information to the corporate network, would you think that you would be met with the same well-practiced and focused response?

As someone who has engaged many organizations both from a computer and a human perspective, I can tell you that you wouldn't. Yet the digital theft could lead to a significantly bigger loss to the organization than the robbery of a single bank branch.

In the words of Tom Peters, "we are not prepared". But we need to be – the clients that I deal with on a daily basis are starting to see that the overwhelming majority of their risk lies with their users, and they are starting to deal with it.

But we are only at the beginning of a new era in information security – one that is far less about the risk presented by the computers and far more reflected in the humans that sit behind the keyboards.

VP Professional Services, CISO Foreground Security
Mike has spent more than a decade helping companies large and small to protect their information by understanding their vulnerability posture from the perspective of an attacker. From his work in the late 90's as a penetration tester and vulnerability researcher to leadership positions at nCircle, Neohapsis and Liberty Mutual Insurance Group, his focus has always been on using vulnerability assessment through penetration testing and social engineering to proactively defend organizations. Mike is currently the CISO of Foreground Security, where he leads engagements to help corporate and government customers understand and protect their security organization. He is also in charge of the advanced curriculum of The Hacker Academy, where he trains security professionals on the newest methods of computer penetration testing and social engineering to help better protect their organizations. Mike's thoughts on security can be found on his blog at Episteme.ca, and his work on helping build careers can be found at InfoSecLeaders.com and ConnectedCareer.com.

© iStockphoto/Yuri_Arcurs

ONLINE TRAINING

English & German (Foundation)

ISTQB® Certified Tester Foundation Level

ISTQB® Certified Tester
Advanced Level Test Manager

ACCREDITED ISTQB TRAINING PROVIDER

ISTQB® International Software Testing Qualifications Board

Díaz Hilterscheid

te testing experience


© iStockphoto.com/KieselUndStein

Software Supply Chain Integrity in SAP Applications

by Sebastian Schinzel, Gunter Bitz, Andreas Wiegenstein, Markus Schumacher & Frederik Weidemann

Today's companies store and process their business assets, or at least critical information related to their business assets in large software systems. Therefore, strong access controls should be incorporated in order to protect those software systems. The access controls should follow the *least privilege* principle: Users should only be able to access the minimal set of information that is required to do their work. As an example, software developers should not be able to access information about salaries of other employees. Therefore, customers have three different systems jn their SAP application landscape: Firstly, developers create a software application on behalf of the customer on a development system. Secondly, the software application is transported to the test system, where software testers validate that the application works properly. Finally, after the software functionality has been validated, the software is transported to the production system. It is only on the production system that the application will process the actual business information. Therefore, neither software developers, nor software testers are able to access critical information as they solely work on the development or testing systems, respectively.

From another viewpoint, developers build the access controls in software applications that have to limit their own capabilities of accessing critical information on the production system. This means that developers must be trusted in two ways:

1. The developer must be trustable in terms of development skill. He should be sufficiently skilled and thorough in order to assure that the implemented access controls are correct and cannot be circumvented. Given the large amount of different security vulnerabilities which allow malicious users to bypass access controls, achieving the required skill set needs a lot of secure development training and experience. Vulnerability lists such as the OWASP Top Ten [3] and the CWE/SANS Top 25 [4] will give you an idea of the large variety of security vulnerabilities in software applications [2].
2. The developer must be trustable as a person. If the developer himself has malicious intentions he may intentionally create flawed access controls in a software application. One possibility could be that he builds a backdoor into an otherwise correctly working access control. Such a backdoor could allow the malicious developer to bypass the access control on the productive application. He could then be able to access critical information without authorization. Generally speaking, companies need to trust employees in order to get work done. This is at least true for internal employees and certain trusted partner companies.

However, the supply chain of software development is growing fast. Companies outsource and offshore software development to external companies, which in turn may outsource parts of the development or use external programming libraries and so forth. The total number of all developers directly involved in the creation of a large software application and the developers of the external components that the software application uses may be in the hundreds or even thousands. Can you trust hundreds or thousands of developers you don't know and who work for companies you may have never heard of?

People usually are not that trusting, which raises the question of how the integrity of the software supply chain can be assured. The Software Assurance Forum for Excellence in Code (SAFECode) created the Software Supply Chain Integrity Framework [1] in order to help companies establish a process to ensure software supply chain integrity. In the following, we will show a practical example of how to test for integrity breaches in software development.

We have seen that malicious developers actually build backdoors into software applications in order to bypass access controls at production systems. However, there is also a good reason for trusted developers to create backdoors: sometimes, development systems or testing systems do not apply the same role and authorization concept as the production system [2]. To allow developers to test the code they have developed with different access rights, they sometimes perform hard-coded authorization checks at the code level. Listing 1 shows what such a check could look like in ABAP, SAP's programming language for business applications [2].

```
1.      IF sy-uname = 'JOHNDOE'.
2.              CALL TRANSACTION 'SM30'.
3.      ELSE.
4.              AUTHORITY-CHECK OBJECT 'S_TCODE'.
5.                      ID 'TCD'.
6.                      FIELD 'SM30'.
7.              IF sy-subrc = 0.
8.                      CALL TRANSACTION 'SM30'.
9.              ELSE.
10.                     show_error_permission_denied(
).
11.             ENDIF.
12.             ENDIF.
```

Listing 1: Example of a backdoor for user 'JOHNDOE'.

In line 1 of listing 1, there is a check for a hard-coded user name. This user is always allowed to call the SAP transaction SM30, which displays sensitive employee data. For all other users, the code correctly performs an authority check starting at line 4. Only if this authority check yields

a positive result, transaction SM30 is called. So the authority check works correctly with the single exception of the user with user name JOHNDOE, who is always allowed to call the transaction directly.

Again note that this code does not necessarily originate from a malicious developer. The developer JOHNDOE could also have included this "hack" solely for the purpose of testing the code on the development or test system. After the testing phase, the developer may have simply forgotten to remove this backdoor before it was deployed to the production system.

In order to prevent this type of backdoor effectively, you should perform several tasks that are distributed over the whole software development process:

1. You should offer regular secure software development training for the developers in order to limit risk with respect to common security vulnerabilities.
2. You should introduce secure coding guidelines that list a set of security requirements that the developers must follow. That way, you prevent lengthy discussions whether an issue is indeed a security vulnerability or just a false positive. They can also be used as an acceptance criterion for external software development projects.
3. External security experts should perform security tests before any custom application is put into production. This effort should not only include a final penetration test, but also code audits of the most critical parts of the application such as access controls and Web front ends. Static code analysis tools [5] have become mature and affordable in the last years. They offer a significant improvement in overall code quality and code security if performed at least on a weekly basis, thus increasing the overall confidence in the integrity of software supply chains.

In summary, the integrity of the growing software supply chains is a serious risk, because more and more people are involved in the creation of software applications. More developers and the increasing complexity of software applications result in a greater risk for security flaws in the applications on one hand. On the other hand, companies create software applications within a long supply chain, which raises the question of how to verify the integrity of the supply chain. A set of security requirements used as acceptance criteria in combination with static code analysis tools increase the confidence in the integrity of software supply chains.

Bibliography

[1]: The Software Supply Chain Integrity Framework, Gunter Bitz et. al., http://www.safecode.org/publications/SAFECode_Supply_Chain0709.pdf

[2]: Sichere ABAP-Programmierung, Andreas Wiegenstein, Markus Schumacher, Sebastian Schinzel, Frederik Weidemann, http://sap-press.de/2037

[3]: OWASP Top Ten Project, http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[4]: CWE/SANS TOP 25 Most Dangerous Programming Errors, http://www.sans.org/top25errors/

[5]: Static Code Security Analysis for ABAP, http://codeprofilers.com/





Sebastian Schinzel has been a developer and security consultant for more than six years in various technology domains. He focuses on application security assessments, as well as secure development of business software applications.

Gunter Bitz is responsible for the Security Governance of SAP's products defines the strategy for testing SAP software.

Andreas Wiegenstein, Chief Technology Officer, Virtual Forge GmbH.
Wiegenstein researches ABAP security and trains developers and security testers at some of the world's largest companies, including SAP.

Dr. Markus Schumacher is CEO of Virtual Forge, a leading company for software security for business applications. He was a representative of the Fraunhofer-Institute SIT. He worked as product manager and project lead at SAP. He holds a PhD in computer science and is frequent speaker at international conferences.

Frederik Weidemann is a Security Consultant with a wide range of project experience in SAP scenarios. He has advanced technical experience in ABAP and Java and offers his knowledge regularly in trainings and presentations.


© iStockphoto.com/DrGrounds

Business Logic Security Testing and Fraud

by James Christie

Is security testing about the technology or the business?

When I started in IT in the 80s, the company for which I worked had a closed network restricted to about 100 company locations with no external connections. Security was divided neatly into physical security, concerned with the protection of the physical assets, and logical security, concerned with the protection of data and applications from abuse or loss.

When applications were built, the focus of security was on internal application security. The arrangements for physical security were a given, and didn't affect individual applications. There were no outsiders to worry about who might gain access, and so long as the common access control software was working there was no need for analysts or designers to worry about unauthorized internal access.

Security for the developers was therefore a matter of ensuring that the application reflected the rules of the business; rules such as segregation of responsibilities, appropriate authorization levels, dual authorization of high-value payments, reconciliation of financial data.

The world quickly changed and relatively simple, private networks isolated from the rest of the world gave way to more open networks with multiple external connections and to web applications.

Security consequently acquired much greater focus. However, it began to seem increasingly detached from the work of developers. Security management and testing became specializations in their own right, and not just an aspect of technical management and support.

We developers and testers continued to build our applications, comforted by the thought that the technical security experts were ensuring that the network perimeter was secure.

Nominally, security testing was a part of non-functional testing. In reality, it had become somewhat detached from conventional testing.

According to the glossary of the British Computer Society's Special Interest Group in Software Testing (BCS SIGIST) [1], security testing determines whether the application meets the specified security requirements.

SIGIST also says that security entails the preservation of confidentiality, integrity and availability of information. Availability means ensuring that authorized users have access to information and associated assets when required. Integrity means safeguarding the accuracy and completeness of information and processing methods. Confidentiality means ensuring that information is accessible only to those authorized to have access.

Penetration testing, and testing the security of the network and infrastructure, are all obviously important, but if you look at security in the round, bearing in mind wider definitions of security (such as SIGIST's), then these activities can't be the whole of security testing.

Some security testing has to consist of routine functional testing that is purely a matter of how the internals of the application work. Security testing that is considered and managed as an exercise external to the development, an exercise that follows the main testing, is necessarily limited. It cannot detect defects that are within the application rather than on the boundary.

Within the application, insecure design features or insecure coding might be detected without any deep understanding of the application's business role. However, like any class of requirements, security requirements will vary from one application to another, depending on the job the application has to do.

If there are control failures that reflect poorly applied or misunderstood business logic, or the business rules, then will we as functional testers detect that? Testers test at the boundaries. Usually we think in terms of boundary values for the data, the boundary of the application or the network boundary with the outside world. Do we pay enough attention to the boundary of what is permissible user behavior? Do we worry enough about abuse by authorized users, employees or outsiders who have passed legitimately through the network and attempt to subvert the application, using it in ways never envisaged by the developers?

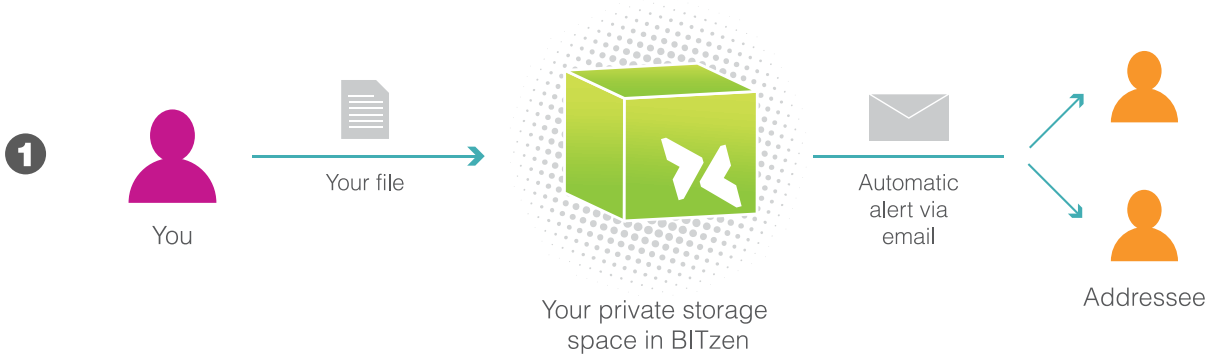
I suspect that we do not, and this must be a matter for concern. A Gartner report of 2005 [2] claimed that 75% of attacks are at the application level, not the network level. The types of threats listed in the report all arise from technical vulnerabilities, such as command injection and buffer overflows.

Such application layer vulnerabilities are obviously serious, and must be addressed. However, I suspect too much attention has been given to them at the expense of vulnerabilities arising from failure to implement business logic correctly. This is my main concern in this article. Such failures can offer great scope for abuse and fraud. Security testing has to be about both the technology and the business.

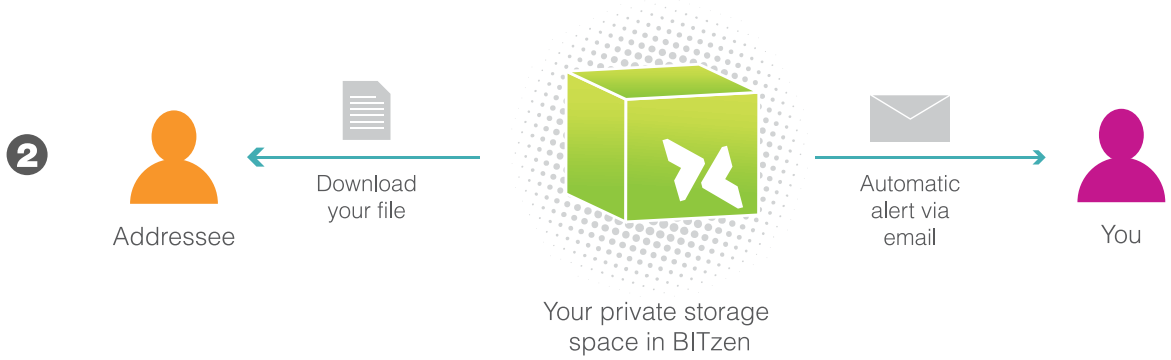
Problem of fraud and insider abuse

It is difficult to come up with reliable figures about fraud because of its very nature. According to PriceWaterhouseCoopers, in 2007 [3] the average loss to fraud by companies worldwide over the two years from 2005 was $2.4 million (their survey being biased towards

Send smoothly...



Upload your file to your private storage space in BITzen.
BITzen inform your addressee via email that they have recieved a new file to download.



Your addressee downloads your file over a secure SSL connection.
BITzen informs you of the successful download.

...with www.bitzen.net

bitzen.net

Exchange files
with customers, suppliers and partners in
a simple, fast and secure way.

- Specifically created for companies and professionals.
- Free account with 100 MB storage space.
- Monthly plans starting from EUR 6.00.
- Paypal.
- Send large files up to 2GB.
- Batch files upload.
- Unlimited contacts.
- Automatic alerts for sent and uploaded files.
- Resend files.
- No additional software installation needed (SaaS).
- No Virtual Disk, no FTP and no Webmail.
- Supports any types of files.
- Compatible with IE, Firefox, Safari, Opera and Chrome.
- Automatic daily backups.
- No advertising. No Spam.
- No public links.
- Unlimited number of downloads.
- Secure SSL.

Web: www.bitzen.net Email: contact@bitzen.net Twitter: @bitzennet

powered by amazon web services

larger companies). This is based on reported fraud, and PWC increased the figure to $3.2 million to allow for unreported frauds.

In addition to the direct costs, there were average indirect costs in the form of management time of $550,000 and substantial unquantifiable costs in terms of damage to the brand, staff morale, reduced share prices and problems with regulators.

PWC stated that 76% of their respondents reported the involvement of an outside party, implying that 24% were purely internal. However, when companies were asked for details on one or two frauds, half of the perpetrators were internal and half external.

It would be interesting to know the relative proportions of frauds (by number and value) which exploited internal applications and customer-facing web applications, but I have not seen any statistics for these.

The U.S. Secret Service and CERT Coordination Center have produced an interesting series of reports on "illicit cyber activity". In their 2004 report on crimes in the US banking and finance sector [4], they reported that in 70% of the cases the insiders had exploited weaknesses in applications, processes or procedures (such as authorized overrides). 78% of the time the perpetrators were authorized users with active accounts, and in 43% of cases they were using their own account and password.

The enduring problem with fraud statistics is that many frauds are not reported, and many more are not even detected. A successful fraud may run for many years without being detected, and may never be detected. A shrewd fraudster will not steal enough money in one go to draw attention to the loss.

I worked on the investigation of an internal fraud at a UK insurance company that had lasted 8 years, as far back as we were able to analyze the data and produce evidence for the police. The perpetrator had raised 555 fraudulent payments, all for less than £5,000 and had stolen £1.1 million by the time that we received an anonymous tip-off.

The control weaknesses related to an abuse of the authorization process, and a failure of the application to deal appropriately with third-party claims payments, which were extremely vulnerable to fraud. These weaknesses would have been present in the original manual process, but the users and developers had not taken the opportunities that a new computer application had offered to introduce more sophisticated controls.

No-one had been negligent or even careless in the design of the application and the surrounding procedures. The trouble was that the requirements had focused on the positive functions of the application, and on replicating the functionality of the previous application, which in turn had been based on the original manual process. There had not been sufficient analysis of how the application could be exploited.

Problem of requirements & negative requirements

Earlier I was careful to talk about failure to implement business logic correctly, rather than implementing requirements. Business logic and requirements will not necessarily be the same.

The requirements are usually written as "*the application must do*" rather than "*the application must not...*". Sometimes the "*must not*" is obvious to the business. It "*goes without saying*" - that dangerous phrase!

However, the developers often lack the deep understanding of business logic that users have, and they design and code only the "*must do*", not even being aware of the implicit corollary, the "*must not*".

As a computer auditor I reviewed a sales application which had a control to ensure that debts couldn't be written off without review by a manager. At the end of each day a report was run to highlight debts that had been cleared without a payment being received. Any discrepancies were highlighted for management action. I noticed that it was possible to overwrite the default of today's date when clearing a debt. Inserting a date in the past meant that the money I'd written off wouldn't appear on any control report. The report for that date had been run already.

When I mentioned this to the users and the teams who built and tested the application, the initial reaction was "*but you're not supposed to do that*", and then they all tried blaming each other. There was a prolonged discussion about the nature of requirements.

The developers were adamant that they'd done nothing wrong, because they'd built the application exactly as specified, and the users were responsible for the requirements.

The testers said they'd tested according to the requirements, and it wasn't their fault.

The users were infuriated at the suggestion that they should have to specify every last little thing that should be obvious - obvious to them anyway.

The reason I was looking at the application, and looking for that particular problem, was because we knew that a close commercial rival had suffered a large fraud when a customer we had in common had bribed an employee of our rival to manipulate the sales control application. As it happened, there was no evidence that the same had happened to us, but clearly we were vulnerable.

Testers should be aware of missing or unspoken requirements, implicit assumptions that have to be challenged and tested. Such assumptions and requirements are a particular problem with security requirements, which is why the simple SIGIST definition of security testing I gave above isn't sufficient – security testing cannot be only about testing the formal security requirements.

However, testers, like developers, are working to tight schedules and budgets. We're always up against the clock. Often there is barely enough time to carry out all the positive testing that is required, never mind thinking through all the negative testing that would be required to prove that missing or unspoken negative requirements have been met.

Fraudsters, on the other hand, have almost unlimited time to get to know the application and see where the weaknesses are. Dishonest users also have the motivation to work out the weaknesses. Even people who are usually honest can be tempted when they realize that there is scope for fraud.

If we don't have enough time to do adequate negative testing to see what weaknesses could be exploited, then at least we should do a quick informal evaluation of the financial sensitivity of the application and alert management, and the internal computer auditors, that there is an element of unquantifiable risk. How comfortable are they with that?

If we can persuade project managers and users that we need enough time to test properly, then what can we do?

CobIT and OWASP

If there is time, there are various techniques that testers can adopt to try and detect potential weaknesses or which we can encourage the developers and users to follow to prevent such weaknesses.

I'd like to concentrate on the CobiT (Control Objectives for Information and related Technology) guidelines for developing and testing secure applications (CobiT 4.1 2007 [5]), and the CobiT IT Assurance Guide [6], and the OWASP (Open Web Application Security Project) Testing Guide [7].

Together, CobiT and OWASP cover the whole range of security testing. They can be used together, CobiT being more concerned with what applications do, and OWASP with how applications work.

They both give useful advice about the internal application controls and functionality that developers and users can follow. They can also be used to provide testers with guidance about test conditions. If the developers and users know that the testers will be consulting these guides, then they have an incentive to ensure that the requirements and build reflect this advice.

CobiT implicitly assumes a traditional, big upfront design, Waterfall approach. Nevertheless, it's still potentially useful for Agile practitioners, and it is possible to map from CobiT to Agile techniques, see Gupta [8].

The two most relevant parts are in the CobiT IT Assurance Guide [6]. This is organized into domains, the most directly relevant being "Acquire and Implement" the solution. This is really for auditors, guiding them through a traditional development, explaining the controls and checks they should be looking for at each stage. It's interesting as a source of ideas, and

as an alternative way of looking at the development, but unless your organization has mandated the developers to follow CobiT, there's no point trying to graft this onto your project.

Of much greater interest are the six CobiT application controls. Whereas the domains are functionally separate and sequential activities, a life-cycle in effect, the application controls are statements of intent that apply to the business area and the application itself. They can be used at any stage of the development. They are;

AC1 Source Data Preparation and Authorization

AC2 Source Data Collection and Entry

AC3 Accuracy, Completeness and Authenticity Checks

AC4 Processing Integrity and Validity

AC5 Output Review, Reconciliation and Error Handling

AC6 Transaction Authentication and Integrity

Each of these controls has stated objectives, and tests that can be made against the requirements, the proposed design and then on the built application. Clearly these are generic statements potentially applicable to any application, but they can serve as a valuable prompt to testers who are willing to adapt them to their own application. They are also a useful introduction for testers to the wider field of business controls.

CobiT rather skates over the question of how the business requirements are defined, but these application controls can serve as a useful basis for validating the requirements.

Unfortunately the CobiT IT Assurance Guide can be downloaded for free only by members of ISACA (Information Systems Audit and Control Association) and costs $165 for non-members to buy. Try your friendly neighborhood Internal Audit department! If they don't have a copy, well maybe they should.

If you are looking for a more constructive and proactive approach to the requirements, then I recommend the Open Web Application Security Project (OWASP) Testing Guide [7]. This is an excellent, accessible document covering the whole range of application security, both technical vulnerabilities and business logic flaws.

It offers good, practical guidance to testers. It also offers a testing framework that is basic, and all the better for that, being simple and practical.

The OWASP testing framework demands early involvement of the testers, and runs from before the start of the project to reviews and testing of live applications.

Phase 1: Before Deployment begins

1A: Review policies and standards

1B: Develop measurement and metrics criteria (ensure traceability)

Phase 2: During definition and design

2A: Review security requirements

2B: Review design and architecture

2C: Create and review UML models

2D: Create and review threat models

Phase 3: During development

3A: Code walkthroughs

3B: Code reviews

Phase 4: During development

4A: Application penetration testing

4B: Configuration management testing

Phase 5: Maintenance and operations

5A: Conduct operational management reviews

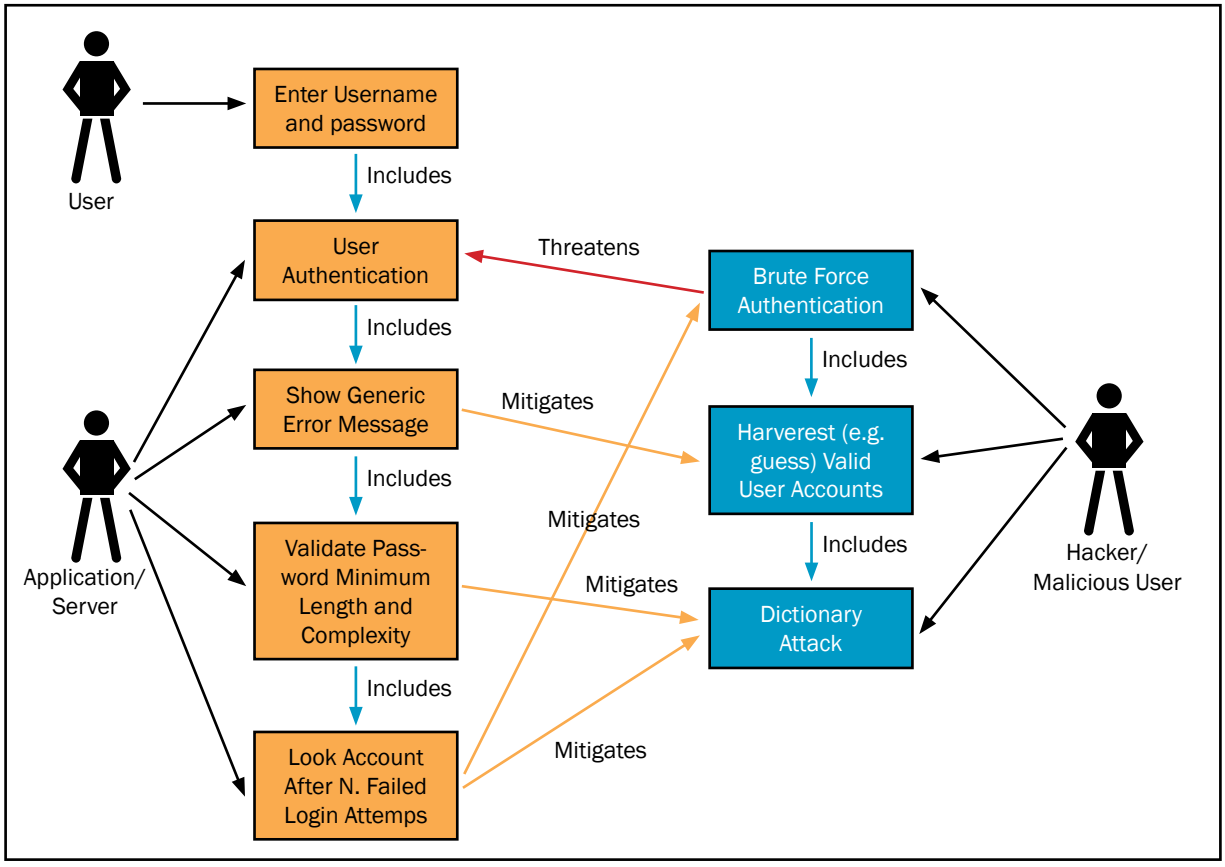
5B: Conduct periodic health checks

5C: Ensure change verification

OWASP suggests four test techniques for security testing; manual inspections and reviews, code reviews, threat modeling and penetration testing. The manual inspections are reviews of design, processes, policies, documentation and even interviewing people; everything except the source code, which is covered by the code reviews.

A feature of OWASP I find particularly interesting is its fairly explicit admission that the security requirements may be missing or inadequate. This is unquestionably a realistic approach, but usually testing models blithely assume that the requirements need tweaking at most.

The response of OWASP is to carry out what looks rather like reverse engineering of the design into the requirements. After the design has been completed, testers should perform UML modeling to derive use cases that "*describe how the application works. In some cases, these may already be available*". Obviously in many cases these will not be available, but the clear implication is that even if they are available, they are unlikely to offer enough infor-



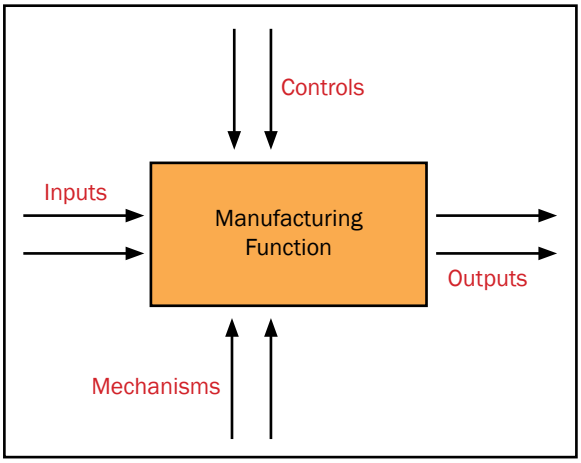
mation to carry out threat modeling.

The feature most likely to be missing is the misuse case. These are the dark side of use cases! As envisaged by OWASP, the misuse cases shadow the use cases, threatening them, then being mitigated by subsequent use cases.

The OWASP framework is not designed to be a checklist to be followed blindly. The important point about using UML is that it permits the tester to decompose and understand the proposed application to the level of detail required for threat modeling, but also with the perspective that threat modeling requires; i.e. what can go wrong? what must we prevent? what could the bad guys get up to?

UML is simply a means to that end, and was probably chosen largely because that is what most developers are likely to be familiar with, and therefore UML diagrams are more likely to be available than other forms of documentation. There was certainly some debate in the OWASP community about what the best means of decomposition might be.

Personally, I have found IDEF0 a valuable means of decomposing applications while working as a computer auditor. Full details of this technique can be found at www.idef.com [9].



It entails decomposing an application using a hierarchical series of diagrams, each of which has between three and six functions. Each function has inputs, which are transformed into outputs, depending on controls and mechanisms.

Is IDEF0 as rigorous and effective as UML? No, I wouldn't argue that. When using IDEF0 we did not define the application in anything like the detail that UML would entail. Its value was in allowing us to develop a quick understanding of the crucial functions and issues, and then ask pertinent questions.

Given that certain inputs must be transformed into certain outputs, what are the controls and mechanisms required to ensure that the right outputs are produced?

In working out what the controls were, or ought to be, we'd run through the mantra that the output had to be accurate, complete, authorized, and timely. "Accurate" and "complete" are obvious. "Authorized" meant that the output must have been created or approved by people with the appropriate level of authority. "Timely" meant that the output must not only arrive in the right place, but at the right time. One could also use the six CobiT application controls as prompts.

In the example I gave above of the debt being written off, I had worked down to the level of detail of "write off a debt" and looked at the controls required to produce the right output, "cancelled debts". I focused on "authorized", "complete" and "timely".

Any sales operator could cancel a debt, but that raised the item for management review. That was fine. The problem was with "complete" and "timely". All write-offs had to be collected for the control report, which was run daily. Was it possible to ensure some write-offs would not appear? Was it possible to over-key the default of the current date? It was possible. If I did so, would the write-off appear on another report? No. The control failure therefore meant that the control report could be easily bypassed.

The testing that I was carrying out had nothing to do with the original requirements. They were of interest, but not really relevant to what I was trying to do. I was trying to think like a dishonest employee, looking for a weakness I could exploit.

The decomposition of the application is the essential first step of threat modeling. Following that, one should analyze the assets for importance, explore possible vulnerabilities and threats, and create mitigation strategies.

I don't want to discuss these in depth. There is plenty of material about threat modeling available. OWASP offers good guidance, [10] and [11]. Microsoft provides some useful advice [12], but its focus is on technical security, whereas OWASP looks at the business logic too. The OWASP testing guide [7] has a section devoted to business logic that serves as a useful introduction.

OWASP's inclusion of mitigation strategies in the version of threat modeling that it advocates for testers is interesting. This is not normally a tester's responsibility. However, considering such strategies is a useful way of planning the testing. What controls or protections should we be testing for? I think it also implicitly acknowledges that the requirements and design may well be flawed, and that threat modeling might not have been carried out in circumstances where it really should have been.

This perception is reinforced by OWASP's advice that testers should ensure that threat models are created as early as possible in the project, and should then be revisited as the application evolves.

What I think is particularly valuable about the application control advice in CobIT and OWASP is that they help us to focus on security as an attribute that can, and must, be built into applications. Security testing then becomes a normal part of functional testing, as well as a specialist technical exercise. Testers must not regard security as an audit concern, with the testing being carried out by quasi-auditors, external to the development.

Getting the auditors on our side

I've had a fairly unusual career in that I've spent several years in each of software development, IT audit, IT security management, project management and test management. I think that gives me a good understanding of each of these roles, and a sympathetic understanding of the problems and pressures associated with them. It's also taught me how they can work together constructively.

In most cases this is obvious, but the odd one out is the IT auditor. They have the reputation of being the hard-nosed suits from head office who come in to bayonet the wounded after a disaster! If that is what they do, then they are being unprofessional and irresponsible. Good auditors should be pro-active and constructive. They will be happy to work with developers, users and testers to help them anticipate and prevent problems.

Auditors will not do your job for you, and they will rarely be able to give you all the answers. They usually have to spread themselves thinly across an organization, inevitably concentrating on the areas with problems and which pose the greatest risk.

They should not be dictating the controls, but good auditors can provide useful advice. They can act as a valuable sounding board, for bouncing ideas off. They can also be used as reinforcements if the testers are coming under irresponsible pressure to restrict the scope of security testing. Good auditors should be the friend of testers, not our enemy. At least you may be able to get access to some useful, but expensive, CobiT material.

Auditors can give you a different perspective and help you ask the right questions, and being able to ask the right questions is much more important than any particular tool or method for testers.

This article tells you something about CobiT and OWASP, and about possible new techniques for approaching testing of security. However, I think the most important lesson is that security testing cannot be a completely separate specialism, and that security testing must also include the exploration of the application's functionality in a skeptical and inquisitive manner, asking the right questions.

Validating the security requirements is important, but so is exposing the unspoken requirements and disproving the invalid assumptions. It is about letting management see what the true state of the application is – just like the rest of testing.

References

[1] British Computer Society's Special Interest Group in Software Testing (BCS SIGIST) glossary. http://www.testingstandards.co.uk/glossary.htm

[2] Gartner Inc. "Now Is the Time for Security at the Application Level", 2005. http://www.sela.co.il/_Uploads/dbsAttachedFiles/GartnerNowIsTheTimeForSecurity.pdf

[3] PriceWaterhouseCoopers, "Economic crime- people, culture and controls. The 4th biennial Global Economic Crime Survey", 2007.

[4] US Secret Service "Insider Threat Study: Illicit Cyber Activity in the Banking and Finance Sector", 2004. http://www.secretservice.gov/ntac/its_report_040820.pdf

[5] CobiT 4.1, IT Governance Institute, 2007. http://www.isaca.org/

[6] CobiT IT Assurance Guide, IT Governance Institute, 2007.http://www.isaca.org/

[7] OWASP Testing Guide, V3.0, 2008. http://www.owasp.org/index.php/Category:OWASP_Testing_Project

[8] Gupta, S. "SOX Compliant Agile Processes", Agile Alliance Conference, Agile 2008. http://submissions.agile2008.org/files/CD-966.pdf

[9] IDEF0 Function Modeling Method. http://www.idef.com/IDEF0.html

[10] OWASP "Threat Modeling", 2007. http://www.owasp.org/index.php/Threat_modeling

[11] OWASP Code Review Guide "Application Threat Modeling", 2009. http://www.owasp.org/index.php/Application_Threat_Modeling

[12] Microsoft," Improving Web Application Security: Threats and Countermeasures", 2003. http://msdn.microsoft.com/en-us/library/ms994921.aspx

James lives in Perth in Scotland . He is currently working as a consultant through his own company, Claro Testing Ltd (www.clarotesting.com).
James has 24 years commercial IT experience, mainly in financial services, with the General Accident insurance company and IBM (working with a range of blue-chip clients) throughout the UK and also in Finland.
His experience covers test management (the full life-cycle from setting the strategy, writing the test plans, supervising execution, through to implementation), information security management, project management, IT audit and systems analysis.
In 2007 he successfully completed an MSc in IT. His specialism was "Integrating usability testing with formal software testing models". He is particularly interested in the improvement of testing processes and how the quality of applications can be improved by incorporating usability engineering and testing techniques.
James is a Chartered IT Professional and a professional member of the British Computer Society (MBCS). He holds the ISEB Practitioner Certificate in Software Testing and is a member of the Usability Professionals Association.


© iStockphoto.com/Inok

A Risk-Based Approach to Improving Software Security

by Rex Black

If you are a software tester, software developer, development or test manager, or another other software professional concerned with quality and security, you probably know that developing secure software is no longer simply desirable—it's completely essential.

Some developers might assume that most security problems arise from the operating system or networking layers, well below the application code they are working on. However, recent figures for Web-based applications from the Open Web Application Security Project (www.owasp.org) show that over three-quarters of security exploits arose from applications (see Table 1).

So, you know you need secure code, but how to get there? What are your security risks? What security failures and bugs do you have? What do these security risks, failures, and bugs mean? How can you reduce security risk in a way that doesn't create new problems? How do you monitor your progress over time? This article will outline seven steps that will allow you to answer these and other questions as you improve your software's security.

Assess the Risks

Applications tend to have characteristic security risks. These risks often arise from the implementation technology. For example, C and C++ are notorious for their lack of inherent array range checking, and consequent buffer-overflow bugs, which allow hackers to insert malicious code into very long input strings. People writing applications with databases have to worry about SQL injection, where hackers put queries into otherwise-benign fields and gain access to sensitive data.

Exploited Vulnerability	Percent Occurrence
Server Applications	41%
Non-Server Applications	36%
Operating System Issues	15%
Hardware Issues	4%
Communication Protocol Issues	2%
Others	2%
Network and Protocol Stack Issues	1%
Encryption Issues	0%

Table 1: Occurrence of Security Exploits by Vulnerability

Security risks can also arise from the business application domain. For example, since they deal in money, banking applications are attractive targets for criminals and a major source of worry for bank IT departments. Applications that store personal information, such as medical history, are subject to regulations like HIPAA that require strict privacy controls.

Risk awareness is the first step in risk reduction. Companies have been reluctant to let outsiders know about the security failures they've had, but some of their failures make the news, and users report others. For example, the Open Web Application Security Project, www.owasp.org, provides good information for those developing Web applications, as does the World Wide Web Consortium's security page, www.w3.org/Security. Carnegie-Mellon's Software Engineering Institute's CERT Coordination Center, www.cert.org, provides a broader look at computer security issues. Last but not least, check out the searchable Risk Digest archives, catless.ncl.ac.uk/Risks, for great anecdotes and commentary on software risks, including security-related risks.

In addition to being aware of the failures, you need to be aware of the underlying bugs themselves. Depending on the kind of applications you're writing, you'll want to read appropriate books and Web sites for hints on common insecure coding constructs, how developers can avoid them, and how testers can find them. For example, entering "secure programming" in the Amazon.com search engine yields dozens of books, some general, some quite specific.

Once you are aware of the kinds of security risks that could affect your software, do a security risk analysis. Identify the specific risk items that you should be aware of. Meet with stakeholders to determine the level of risk in terms of likelihood and impact. Likelihood relates to the chances of any given risk becoming an actual security bug in your software. Impact relates to the effect on customers, users, and your software should the bug be exploited. Your analysis of the risks and their associated levels of risk will allow you to create a prioritized list of potential security failures. [1]

Test to Know Where You Stand

If you're like most software development organizations, you don't have the luxury of starting over with new code on every project. How secure is that collection of existing code? If you're like many organizations, you haven't really had a chance to check. So, check the security of your existing software through a security test.

1 I describe the process of risk analysis in my books Managing the Testing Process and Pragmatic Software Testing.

This type of test is often called a penetration test. Its purpose, as the name suggests, is to discover ways in which hackers and other unauthorized users can penetrate your system. Such a test is useful to check for security failures that your application already presents to the real world.

Remember that the best lock in the world does no good if it's installed in a door made of rotten wood. Similarly, applications with great security features that are installed in insecurely-configured environments can be hacked.

Do your installation procedures, user documentation, provisioning processes, and notification mechanisms support or impede security? I recently signed up for an account on an e-commerce site that seemed to have good security at first. I was asked to create a user name and password. The application enabled SSL encryption during this process. The input field masked the password when I entered it. I was then told that the application would e-mail me an activation notice after it verified my information. When I received the activation notice, the user name and password were in the e-mail, unencrypted and available to anyone who saw or intercepted that e-mail! Private and identifying information should not be stored or transmitted in an insecure fashion.

Consider identifying risk cases for each security requirement. Risk cases are like use cases—though perhaps more properly termed "misuse cases"—that lay out various scenarios of security failure. If you think about end-to-end processes that users go through, along with the environments in which your software will be deployed, you may think of some possible failures or issues you otherwise would have missed. You can confirm the presence or absence of these failures through specific tests.

You can learn how to run penetration tests yourself. Alternatively, you can hire a testing services provider to handle it for you. On the one hand, you might have to make a significant investment in training and books to learn how to perform penetration tests properly and therefore decide a professional external resource can do a better job. On the other hand, you might feel more comfortable having security expertise in your team and therefore decide to invest in growing it.

Thoroughly testing applications that will run in various installed environments can be a real challenge. Such tests are a combination of end-to-end process testing, compatibility testing, and penetration testing. Depending on the multiplicity of environments, users, and procedures that your application can support, such tests cost a lot of money in terms of systems and effort. To save money on setting up a large variety of test configurations in-house, consider using a testing service provider.

Your prioritized list of risks should guide the penetration test, but you should also test for other failures that you might not have thought of. Based on the failures you find, revise your list of risks. Add new risks where you find unexpected failures. Increase the likelihood and impact based on the failures you find. You might also decrease the likelihood and impact for risks that don't relate to observed failures, or relate to failures that were less important than you expected. However, be careful about assuming that a risk that isn't exploitable today won't be exploitable in future releases of the software.

Keep a list of the security problems you find and where you found them. You'll need this list to fix the problems, of course. However, I also recommend that you classify the problems in a few ways. One classification is based on the type of security flaw.[2] Another is the date on which the code was written or the version of the software in which it was introduced. Yet another is the major subsystem or component the code is part of. In addition, classify the severity (impact on the system) and priority (impact on the user) of each failure. Finally, classify each problem based on the security risks you identified earlier.

Analyze to Know Where You Stand

The security test mentioned above will find security-related failures. However, not every security bug in the code will always exhibit a security failure. In other words, it is possible to have underlying bugs that did not exhibit any symptoms during the penetration test. Therefore, to find additional problems, do a static analysis of the code.

Static analysis means going through your code to look for bugs that could cause failures. You might have input fields which are not appropriately checked for size or syntax before being handed off for processing. You might have weak error handling. You might have situations where unauthorized users can pass snippets of languages like SQL or Korn shell into the system where they would be executed. Just because these bugs didn't result in failures doesn't mean they aren't bugs, and you should look for them.

You can automate your static analysis using tools. A wide variety of static analysis tools for identifying security weaknesses in code exist, so you can probably pick one that fits your exact language, environment, needs, and budget. For a large, existing code base, these tools will identify a large number of problems. Not all of these problems are of the same severity and importance. Somehow, you'll need to focus your attention on the most important of them. Fortunately, good tools will allow you to turn particular rules on and off and tune your static analysis at a level of granularity as fine as individual lines of code. Again, your list of risks can help guide you as you determine where to focus.

Based on your static analysis, add to your list of security problems where you found each problem, and its classification.

Evaluate to Understand Where You Stand

You've gathered a lot of data in the first few steps. Time to evaluate that data. What does the data mean; i.e., what information and patterns are hiding in the data? What is a smart plan of action for improving software security?

First of all, sort the problem list by priority and severity. You will likely want to immediately fix the problems with the highest levels of priority and severity. Microsoft famously reached a point where the number of critical security bugs became so high that they embarked on a crash problem to resolve these bugs. For months, Microsoft programmers did nothing but address security bugs. You might not be in as deep a hole—or be able to spare that much effort—but you'll want to address the urgent items right away.

However, you should also do some further evaluation before wading into battle with the security bugs. Bugs do not tend to be evenly distributed across the code base, but rather tend to exist in clusters. Decades ago, IBM studied their MVS software and found that 38% of the bugs that caused problems in production lived in 4% of the modules. On an Internet appliance project, I found that 69% of the bugs we discovered during testing lived in 25% of the modules. By looking for modules with particularly high numbers of security bugs, you might find that completely refactoring one or two modules is the smartest way to improve your software's security.

As you start to think about the long-term, evaluate how many bugs arise from each kind of security flaw. This will tell you the most typical problems that you and your team face. Can you reduce the incidence of such problems through training for your developers? Better code reviews? Better design reviews? All three? After all, you don't want to be fighting a constant battle against security problems with every release, so you and your team need to learn how to create better software.

You should also evaluate the incidence of security bugs based on the age of the code in which they were found. Software tends to wear out over the years, not as physical devices do, but rather through on-going maintenance which reduces the quality of the code. In addition, older code that was written when a programming language was new—or when the team was new to the language or technology—might contain more bugs. Plan for long-term refactoring of decrepit modules that are disproportionate contributors to software insecurity.

Repair the Problems - Carefully

Any time a developer repairs a bug in software, there is a risk that might introduce a new bug. Many people call these regression bugs, because they represent some reduction in the level of software quality that was

2 For example, you can use the OWASP's Top Ten Web application security flaws if you are creating Web applications (www.owasp.org/index.php/Category:OWASP_Top_Ten_Project).

present before.

The risk of regression bugs applies to security bugs as much as any other bug. In addition, you can't assume that repairing a security bug would necessarily introduce either no bug at all or another security bug. Fixing a security bug might introduce a functionality bug. So, as you repair the security bugs, make sure you have a plan to deal with regression risk. How can you do so?

Your test team typically deals with part of the regression problem. They might have created an automated suite of regression tests for functionality, performance, reliability, or other important quality characteristics. However, waiting for the end-stage testing is not ideal, as the cost and schedule implications of dealing with a bug increase the longer that bug is in the system.[3]

So, before delivering code to the test team, the developers should use code reviews, static analysis, and automated unit tests to help manage regression risk for each change they had made to the system. Code reviews, ideally performed by at least two experts in addition to the author, should help catch many problems. Using the static analysis tool you've already invested in to check your new code is a best practice, and good static analysis tools can find many types of problems, not just security problems. Finally, the use of an automated unit testing harness will provide a framework for an automated set of tests that will allow developers to modify and refactor code with a higher level of confidence.[4]

Examine Results in the Real World

Any time you make a process change, you should monitor how those process changes affect the real world. For example, a couple years ago I was training for a marathon, but I hurt my ankle by overtraining in hills. So, I switched to a training schedule that focused on low-impact aerobic exercises like bicycling and elliptical machines while my ankle healed. Did this process change help me achieve success? Two real world measures applied:

1. Based on the symptoms in my ankle, did it heal while I continued this training regimen, and could I gradually reintroduce running to the training? The answers to both questions were "yes."
2. Was I actually able to run the marathon without pain and without re-injuring myself? Thankfully, the answer to this question was "yes" as well.

Similarly, you want to make sure that your new process reduces the number of known security bugs in your code over time, that the test team finds fewer bugs during system test execution, and that the number of security-related incidents that occur in the field gradually goes down.

You should not expect that these three numbers would go down monotonically. Some natural variation in the testing and development processes will mean the number of known bugs might go both up and down. However, the trend over the long-term (say, one year or more) should be that the average number of known security bugs in any given month has gone down.

Similarly, you might have good months and bad months—months where no field security incidents are reported and months where a rash of them are—but this might simply be a natural variation in usage patterns or seasonal usage. For example, you would expect that financial application security bugs related to fiscal-year closing operations would increase at the end of the year. However, again, the trend over the long-term should be that the average number of security incidents in any given month has gone down.

In addition to monitoring your own security bugs and failures, follow the news. The Internet and trade magazines can help you check for problems in applications similar to yours in business domain, implementation technology, or both. If you hear stories about problems that you think might constitute a risk for your application, update your risk analysis and re-evaluate accordingly.

Institutionalize Success

The last step of this process is to do everything all over again, on every single project. That's something of an overstatement, since you don't need to start from a clean slate. You will need to repeat process, though, using your existing work as a baseline:

1. Re-assess security risks.
2. Re-test the application for security failures.
3. Re-analyze the software for security bugs.
4. Re-evaluate patterns in security risks, failures, and bugs.
5. Repair with care.
6. Re-examine the real-world results.

In each of these steps, make sure you look both at new concerns related to changes to your applications and concerns you might have previously overlooked.

Institutionalizing success, the final step of process improvement, is very easy to overlook. After a big push to improve software security, you might be tempted to celebrate success, relax your guard, and gradually slip back into old practices of coding.

We recently had a client that asked us to run a penetration test of their systems. We found a number of security failures during this test, and reported our findings to the engineering team. Later in the project, shortly before release, we re-ran the penetration test. The engineers had resolved all of the failures we had found previously. However, they had also built a bunch of new stuff, which had the exact same kinds of underlying security bugs exhibiting similar security failures. My client had dealt with the manifestations of bad security practices by repairing the security bugs we had found the first time, but had not changed the bad security practices themselves.

Conclusions

Software security is an important concern, and it's not just for operating system and network vendors. If you're working at the application layer, your code is a target. In fact, the trend in software security exploits is away from massive, blunt-force attacks on the Internet or IT infrastructure and towards carefully crafted, criminal attacks on specific applications to achieve specific damage, often economic.

In this article, I laid out a seven-step process to reduce your software's exposure to these attacks.

1. Assess security risks to focus your improvements.
2. Test the software for security failures.
3. Analyze the software for security bugs.
4. Evaluate patterns in security risks, failures, and bugs.
5. Repair the bugs with due care for regression.
6. Examine the real-world results by monitoring important security metrics.
7. Institutionalize the successful process improvements.

Carefully following this process will allow your organization to improve your software security in a way which is risk-based, thoroughly tested, data-driven, prudent, and continually re-aligned with real-world results.

3 For more on the economics of defects, see my article "Testing ROI: What IT Managers Should Know," on the Library page of our Web site, www.rbcs-us.com.

4 For a detailed case study of how RBCS helped one client implement a process of code reviews, static analysis, and automated unit testing, including creation of an automated test tool framework, see my article "Mission Made Possible," written with Greg Kubaczkowsi, at the Library page of our Web site, www.rbcs-us.com.

With a quarter-century of software and systems engineering experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For over a dozen years, RBCS has delivered services in consulting, outsourcing and training for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups and hires testing staff for hundreds of clients worldwide. Ranging from Fortune 20 companies to start-ups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation and more. As the leader of RBCS, Rex is the most prolific author practicing in the field of software testing today. His popular first book, Managing the Testing Process, has sold over 35,000 copies around the world, including Japanese, Chinese, and Indian releases. His five other books on testing, Advanced Software Testing: Volume I, Advanced Software Testing: Volume II, Critical Testing Processes, Foundations of Software Testing, and Pragmatic Software Testing, have also sold tens of thousands of copies, including Hebrew, Indian, Chinese, Japanese and Russian editions. He has written over thirty articles, presented hundreds of papers, workshops, and seminars, and given about thirty keynote speeches at conferences and events around the world. Rex has been President of the International Software Testing Qualifications Board and is a Director of the American Software Testing Qualifications Board.

© Katrin Schülke

k a n z l e i

h i l t e r s c h e i d

Berlin, Germany

IT Law
Contract Law

German
English
French
Spanish

www.kanzlei-hilterscheid.de
info@kanzlei-hilterscheid.de

© iStockphoto.com/bunhill

Demystifying Web Application Security Landscape

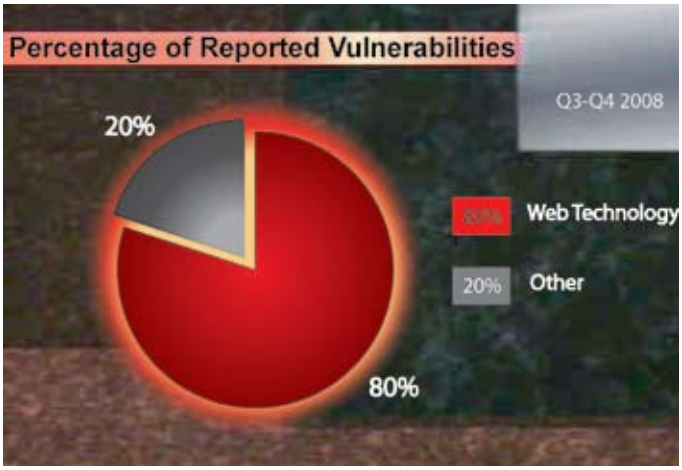
by Mandeep Khera

U.S. Government passes the stimulus package and includes $355M for cyber security. Hacking against Government sites including electric grid intensifies. New regulations for privacy are being passed or proposed across Europe, Asia, and Americas (http://lastwatchdog.com/senate-bill-mandates-strong-federal-role-internet/). Hacking against government agencies and corporations across the globe continues at a faster rate than ever. From social networking sites like Facebook and Twitter to large corporations like Heartland to government agencies like Utilities and Pentagon – no one has been spared. Even countries are using cyber wars as the latest lethal weapons against one another. A lot of these attacks are occurring at the Web application layer meaning through the Web sites.

Hackers are getting smarter. They are no longer trying to attack the network layer which has gotten a lot more secure over the last few years with a vast majority of organizations deploying firewalls and Intrusion Detection Systems (IDSs). The low-hanging fruit for hackers are Web applications. Why? Because that's where the vulnerabilities are. Our research shows that Web application vulnerabilities continue to dominate amongst the total published vulnerabilities, as is evidenced from the chart below.

In spite of all the attacks, regulations, and all the hype, why aren't companies and governments doing something about it? One reason is that most people still don't quite understand what Web application security means or they don't believe they'll ever get hacked. In this article, we are going to try and demystify the application security landscape and also bust some of the myths around this space.

Before we look at Web application security, let's define a Web application. Most consumers and even a lot of IT professionals don't realize that Web sites that allow us to do business transactions online are powered by Web applications. And, in some cases there are hundreds or even thousands of these applications acting as the engine for Web sites. Simply put, a Web application is a software program that's written in a browser- supported language like HTML, and is accessed over the browser. This is different from the older ways of applications that used to have fat clients which accessed the server.



Source: Cenzic Application Security Trends Report – Q3-Q4, 2008

Technically, the fact that these Web applications are vulnerable is not a new phenomenon. The fact is that they have always been vulnerable since the early days of the Web in the late 90s. However, hackers started exploiting these vulnerabilities in the early 2000s as networks got more secure and hackers realized that most Web sites were wide open for hacking.

So, how did these applications get deployed with so many vulnerabilities? First, most developers were not trained to think about security when developing applications. During the client-server era, there weren't any public facing applications, so no one thought it would be a big issue. The Web made everything open which was great for business, but with everything good comes something bad. In this case, we got the hackers. Secondly, even developers, who were trained in security and really understand security issues, suffer from lack of time. Most developers are extremely busy and under a time crunch to get the applications out in time. They don't have time to even do unit testing, forget about security testing.

There are a lot of vendors touting Web application security since it's become the buzz phrase of the decade. At a high level, there are really only two broad categories of solutions in this space as follows:

- **Web Application Security Vulnerability Management** – One can view this as testing of Web applications just like testing of applications for functional and performance features. There are a number of options available to do this type of testing including manual pen testers – internal or external, Web application scanners (black box or dynamic testing) done through the Web interface just like a hacker, Web application scanning in Software as a Service/ managed service, source code scanning (scanning of raw code to find vulnerabilities). Most experts and analysts agree that the best approach to start with is dynamic or black box testing, because you get the biggest bang for the buck. If you do not have expertise in-house, a SaaS solution is the best one to start with. Some vendors like Cenzic offer both Software and SaaS solutions to help customers transition from one to the other if appropriate. Manual pen testers can add further value by helping customers with the processes and remediation information as well as address security holes that cannot be found through automation like social engineering types of vulnerabilities.
- **Web application firewall** – Web application firewalls are similar to network firewalls, except that they focus on the application layer (layer 7 in the OSI model) and can block traffic based on known vulnerabilities if configured properly. Firewalls are either in software or appliance formats. WAFs can be good for cases where companies don't have the time to fix all the vulnerabilities in time. However, WAFs do require proper configuration of rules or they might have false positives and block good traffic, which can be devastating for a business.

File: test_reports_and_screenshots.zip Size:1.67 GB

100%

Recipients: John M., Edward J.

SEND COMPLETED!!!

Send smoothly with
www.bitzen.net

bitzen.net

Exchange files
with customers, suppliers and partners in
a simple, fast and secure way.

Specifically created for companies and professionals.
Plan free.
Monthly plans starting from EUR 6.00
Paypal.
Send large files up to 2GB.
Batch files upload.
Unlimited contacts.
Automatic alerts for sent and uploaded files.
Resend files.

No additional software installation needed (SaaS).
No Virtual Disk, no FTP and no Webmail.
Supports any kinds of files.
Compatible with IE, Firefox, Safari, Opera and Chrome.
Automatic daily backups.
No advertising. No Spam.
No public links.
Unlimited number of downloads.
Secure SSL.

Web: www.bitzen.net Email: contact@bitzen.net Twitter: @bitzennet

powered by amazon web services

All the other technologies that are touted as Web application security such as encryption, identity management, PKI certificates, etc. are necessary technologies in their own right, but should not be confused as Web application security.

Besides technology, organizations need to make sure that they have good processes in place and that proper accountability of application security is clear throughout the enterprise. If InfoSec is responsible for testing for vulnerabilities, clear rules need to be established for developers to fix the vulnerabilities based on priorities. Web application security is not a one time event, and it needs to have an ongoing focus with continuous testing and remediation.

Even with all the hype around Web application security and the various solutions available, most companies are not doing anything or enough about it. Besides pure inertia, there are many myths that are holding companies back from moving forward on these initiatives. Here are the 10 common myths:

- *I have SSL, so my Web sites are secure*: Well, Secure Socket Layer (SSL) has its place in helping provide some protection to the consumers while they are conducting transactions online. However, it does nothing to protect hackers from hacking into Web sites. SSL is a good technology that assures the consumer that the server he/she is connecting to is a valid server, and it encrypts the communication between the browser and the server. So, the SSL lock symbols on most of the sites can be misleading.
- *I have a network firewall, so that should be enough*: Network firewalls were created to keep the bad guys out at the network layer. They were not designed to protect traffic at the Web application layer. NW Firewalls do not validate user inputs to an HTML application, and they don't detect maliciously modified parameters in a URL request.
- *I have an Intrusion Prevention System, so that should prevent me from all intrusions*: IDS and IPS are good devices, but again at the network layer. These devices were created to block access and do not understand injections at the Web application level. Hackers have access to the same forms and fields that consumers do, and unless you use IPS to block all your Web traffic, it's not going to help you thwart hacker attacks.
- *My data is encrypted, so why should I care if someone attacks*: While data encryption is a solid best practice to follow, it's not enough to protect Web applications. Once hackers come in through the Web site and are able to exploit the application, they can access the data and decrypt later.
- *I don't have any public-facing Web applications*: Insider threats are bigger than ever. Even if you don't have external-facing applications, you still have to protect your employee data on internal applications.
- *I can test my Web application once a year and I'll be fine*: Every month there are 400+ new application-related vulnerabilities and hackers know about them. Also, every time you make any change to a Web application, you have to make sure that there are no new vulnerabilities. Application testing has to be a continuous process with at least monthly if not weekly testing.
- *I have never been hacked, so I am fine*: First of all, gone are the days when hackers used to hack to gain fame. Now, most Web hacking is done by organized criminals and in some cases by government-sponsored organizations. These guys don't want you to know that you are being hacked. There are more and more companies who are finding out that they were being hacked for over a year before they discovered the attacks. Also, do you really want to take a chance to see if you get hacked?
- *I only have commercial applications from large vendors, so it's not my problem*: You are responsible for all the customer and employee information, even if you didn't create the application. You need to make sure that you test all the applications as well as any plug-ins that you wrote for those commercial applications. Notify your commercial vendor and put pressure on them to release a patch so you have recourse in case of a breach.
- *I have never been audited* – You have to protect your Web applications to secure your most important asset – customer information. If your applications are secure, you'll pass the audit and comply with regulations. The reverse is not necessarily true.
- *Application Security is painful to implement* – Although it's more difficult to secure Web applications than the network layer and desktops, there are many easy solutions to get your process jumpstarted. Like all initiatives, once you get going, the road gets less bumpy.

All indications are that cyber attacks at the Web application layer will continue to rise in the coming months and years. With 80% of vulnerabilities in Web applications and over 75% of attacks happening through the Web sites, the question is not IF you will get attacked, the question is WHEN you will get attacked.

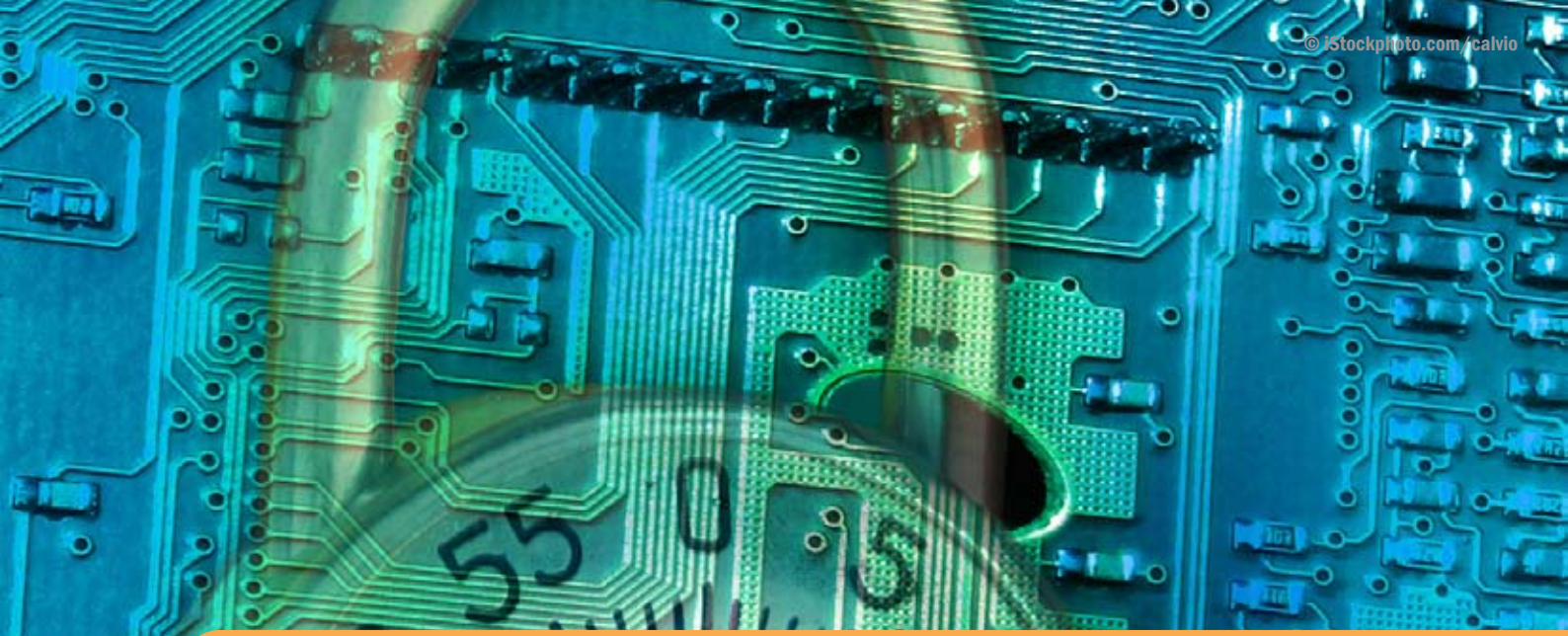
There are a lot of good resources available to help you get going with your initiatives including OWASP, SANS, and many other free Webinars from various vendors like Cenzic. There's a lot of help available to move you along the process. You need to take that first step.

Mandeep Khera has over 24 years of diversified experience in marketing, engineering, business development, sales, customer services, finance and general management.

Mandeep has been with Cenzic, an application security software company, since 2003 as the Chief Marketing Officer driving the strategic, product, and outbound marketing functions. Prior to joining Cenzic, Mandeep managed marketing functions for VeriSign's Managed Security Services product line, as the Vice-President of Marketing for Maaya, a Web-Services Software company, as Vice-President of Worldwide Marketing for UCMS, an eCRM software company and as the Vice-President of Marketing and Engineering for Embarcadero Systems Corporation, a logistics management software company.

Previously he was with Hewlett-Packard for 11 eleven years in various positions including as a general manager of a software business unit.

Mandeep is a graduate of Harvard Business School's Leading Product Development program and Northwestern University's Executive Development Program. He holds a Bachelor of Commerce with honors from New Delhi.


© iStockphoto.com / calvio

Security Testing by Methodology: the OSSTMM

by Simon Wepfer & Pete Herzog

Security tests are an important part of the risk management process and executives realize the benefits of an independent security test: It introduces a neutral view on the target and can improve security when the proposed sensible measures are successfully applied. But there are often also questions to answer after such an audit.

How secure is the target, and are there aspects that have not been tested? How much has our security improved since the last test? How does our security compare to other companies in our industry? This article is a brief introduction into the Open Source Security Testing Methodology Manual (OSSTMM), which can answer these and other follow-up questions.

OSSTMM is a freely available manual that provides a methodology for a thorough security test of physical, human (processes) and communication systems. A core aspect are the security metrics – the Risk Assessment Values (RAV) – which express the final security level of the tested system as a numerical value. The current release candidate of OSSTMM 3.0 is an approximately 150-page document and is a complete re-write from the 2.X version series incorporating the results of the last 6 years of research.

The main purpose of the OSSTMM is to provide a scientific methodology for the accurate characterization of operational security and is adaptable for penetration tests, ethical hacking, security assessments and so forth. In the EU-sponsored project, Open TC, it became the standard for testing and measuring trusted computing systems. Most of all, an OSSTMM compliant test defines the target clearly, and results are reproducible, something unusual in the current methods of ethical hacking and penetration testing.

Preparation and Testing

Before the test can actually start, the assets that have to be secured must be defined. The protection mechanism for the assets are the targets to test. The engagement zone is the area around the assets, the test scope is everything needed to keep the assets operational, for instance, processes or network protocols. The test vector defines the interaction points of the scope. For instance, a DMZ may be tested from the internet or from the LAN as well – with obviously different results. Then, the testing channels have to be defined. Our example DMZ may be tested not only on the communication layer but on the process layer as well (e.g. patching process).

The test type defines the knowledge about the target and the test. Common known testing types are black box and white box; the OSSTMM, however, distinguishes six types, each detailing different results. The rules of engagement are protecting the customer and the tester on legal, ethical and procedural aspects.

When all the above has been defined well, it is clear which tests in the OSSTMM have to be performed on the scope. OSSTMM tests define

Classification	Description
Vulnerability	is the flaw or error that: (a) denies access to assets for authorized people or processes, (b) allows for privileged access to assets to unauthorized people or processes, or (c) allows unauthorized people or processes to hide assets or themselves within the scope
Weakness	is the flaw or error that disrupts, reduces, abuses, or nullifies specifically the effects of the five interactivity controls: authentication, indemnification, resistance, subjugation, and continuity.
Concern	is the flaw or error that disrupts, reduces, abuses, or nullifies the effects of the flow or execution of the five process controls: non-repudiation, confidentiality, privacy, integrity, and alarm.
Exposure	is an unjustifiable action, flaw, or error that provides direct or indirect visibility of targets or assets within the chosen scope channel.
Anomaly	is any unidentifiable or unknown element which has not been controlled and cannot be accounted for in normal operations.

only what is to be done, but do not dictate any tools. One test for data networks for example is requesting that server uptime has to be verified to latest vulnerabilities and patch releases. Another example is that responses to UDP packets with bad checksums to a collection of ports have to be verified. The tools to use and how to use them is left up to the tester.

An OSSTMM compliant test is much more than running an automated vulnerability scanner and printing the report. It relies on the tester's in-depth knowledge and experience, and on human intelligence for interpreting the results. This does not mean that automated tools will not be used at all, but they will be used as what they are: just a tool without real intelligence.

Risk Assessment Value

Once a risk is detected and verified, it has to be categorized. OSSTMM is naming these limitations; the inability of protection mechanisms to work correctly, see table

OSSTMM knows five „risk" classifications

The limitations are one of the three factors for calculating the final RAV. The operational security is a second one, derived from visibility (a means of calculating opportunity for an attack), access (counting the interactive access points) and trust (fall-back to unauthenticated access to trusted systems). The third factor for calculating the RAV are the controls implemented for each point identified in the operational security section. Controls are grouped in class A (authentication, indemnification, subjugation, continuity and resilience) and class B (non-repudiation, confidentiality, privacy, integrity and alarm).

Certification

ISECOM (Institute for Security and Open Methodologies) offers several OSSTMM-specific certification and training schemes. The ISECOM Licensed Auditor (ILA) program provides quality assurance and support for obtaining OSSTMM certified audits from a properly accredited auditing company. OPST (OSSTMM Professional Security Tester) and OPSA (... Analyst) is a certification for persons. Additional information may be found on http://www.isecom.org/.

Simon Wepfer is COO at OneConsult.

Pete Herzog is founder of the OSSTMM and Director of ISECOM.


© iStockphoto.com/alexsl

Application Security Fundamentals

by Joel Scambray

As the importance of security continues to dawn on the Software Industry 2.0, organizations of all sizes are trying to discover what constitutes software security "due care" for their customers. This brief paper will review key principles surrounding security in the development lifecycle (SDL), covering economic drivers, industry benchmarks (or the lack thereof), a prototypical SDL model, and what we've seen work/not work with real-world SDL implementations.

Few question that software occupies an increasingly central role in our everyday lives. From computer operating systems and applications, to mobile phones, television, the Internet, VoIP, GPS navigation, software-driven medical systems, air traffic control, the electric grid, and so on, human activity (and perhaps even human existence itself) has come to rely heavily on software.

But is that reliance justified? As more and more of our lives becomes digitized, and headlines have begun to trumpet the growth of malicious hacking and other incidents of cyber abuse, deep questions surrounding the confidentiality, integrity, and availability[1] of digital data have been raised. Are the risks underlying this brave new world greater than the rewards?

In parallel, software development organizations of all sizes are trying to discover how to protect their end users from unreasonable risks. Of course, this raises yet another question: what constitutes "reasonable"? Put another way, what is the standard of software security "due care"?

To date, the software industry has adopted the following fundamental approaches to establishing this standard:

1. Ignore
2. React
3. Prevent

Let's examine each of these approaches briefly to set the stage for a deeper discussion of security in the software development lifecycle (SDL).

Ignorance is Bliss

The dirty little non-secret of the technology industry is that few software development-oriented companies are doing anything serious about security. Recent surveys suggest that, despite some uptake of outsourcing and tools, most firms do not allocate significant budget or headcount for application security outside of standard operational IT security processes [1]. Although some in the information security industry would bristle at the implication, the question remains: Is ignoring the problem simply good risk management?

1 Confidentiality, integrity, and availability (CIA) are often cited as the defining properties of information security. Some authorities also include a fourth "A" for "accountability," typically understood to refer to the keeping of tamper-resistant activity logs to provide non-repudiation.

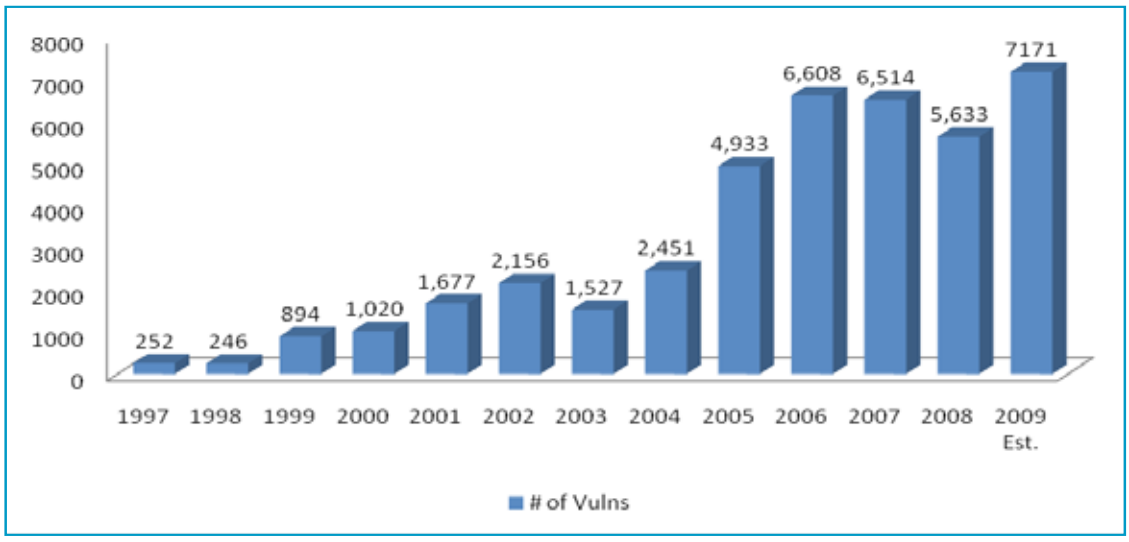


Figure 1: Software vulnerabilities released between 1997 and 2009 (extrapolated), courtesy of National Vulnerability Database

Data on security incidents or breaches has historically not been tracked systematically (with some recent exceptions, albeit focused on operational breaches rather than purely software vulnerability-related [2]). There is good news and bad news in this recent data: while only 6 out of 90 confirmed breaches (derived from over 150 cases) "…resulted from an attack exploiting a patchable vulnerability… the bulk of attacks continues to target applications and services rather than the operating systems or platforms on which they run." (ibid) So, if you're a major operating system vendor, take heart, but if you're writing custom application code, you're increasingly the target of attack.

Another quasi-informative dataset is the National Vulnerability Database (NVD), which tracks advisories on software vendor bulletins [3]. Figure 1 shows the raw count of vulnerabilities tracked in NVD, across all vendors and products, including software flaws (not configuration flaws), across all sources, from 1997 to 2009.

Clearly, the number of visible vulnerabilities is on the rise. Presumptively, based on unrelated studies showing software sales to be dominated by a handful of vendors, most of these vulnerabilities emanate from the same group of vendors. (We haven't done research confirming this.) Given all this activity, does it make sense simply to "vanish in the noise" if you are a small or medium sized software shop, and simply invest minimally in, say, outsourced professional security review during the release cycle? What is the return on investment (ROI) for security in the development lifecycle? We'll return to this question later, but will simply note at this point that there is a robust world-wide research community waiting for you out there.

To close this discussion of the merits of ignoring the software security problem, although it seems counterintuitive to assert positive outcomes from such a mindset, we've found minimal data to quantify the penalties of such an approach for small- to mid-sized software development efforts. Larger-scale development organizations with widely-deployed products are a different story, and anecdotal evidence exists to support investment in security, a topic we'll return to later. Next, we'll examine the other two postures, reactive and preventive.

React vs. Prevent

Many software firms have observed real-world security quality improvements resulting from external security review, and have hired penetration testers to assess their products, typically keeping the results private and selectively fixing some or all of the vulnerabilities. Although these practices can be laudable when performed in conjunction with other measures to be discussed momentarily, simply finding and fixing bugs iteratively between releases is not necessarily the most efficient way to increase code security quality. In fact, it's arguably less efficient than "learning to fish", in other words adapting a culture of prevention and the processes & technology to support it.

This is the heart of the justification for SDL. "Baking security in" rather than "bolting it on" in theory leads to better outcomes for all involved, including the development organization and its customers. Next, we'll describe in more detail the components of SDL and how it drives these outcomes.

SDL History and Philosophy

Of course, the notion of "baking security in" has been around for some time. Some of the "classic" antecedents of security in the development lifecycle include:

- NIST SP 800-64 [4]
- BS7799/ISO17799/27001-2 [5]
- OCTAVE [6]

In our opinion, ISO 17799 Sec 10 and ISO 27002 Sec 12 remain classics from an SDL policy perspective, including such fundamentals as separation of test and production environments, input/output validation, cryptography best practices, transaction integrity/non-repudiation, and so on.

More recent iterations of security in the development lifecycle frameworks include:

- Microsoft SDL [7]
- CLASP [8]
- BSI-MM [9]
- OpenSAMM [10]

Microsoft's SDL is among the most widely recognized currently, although there has been substantial recent attention for the other frameworks in this list (which share some overlaps in pedigree [11]).

SDL Principles & Framework

Obviously, there are a number of approaches to security in the development lifecycle, going back several years. It is therefore more realistic to think of SDL as a framework, or set of principles, that specific organizations can adapt and customize to their own unique purposes. Even Microsoft's SDL documents refer to their "implementation" of SDL. Below we attempt to summarize some of the high-level principles of SDL that are common to many of the above-mentioned frameworks:

1. Distribute the work of assessment and remediation, especially to the development team
2. Independent (of the development team) reviews at key milestones
3. Provide relevant training and re-usable guidance (checklists)
4. Strive for quantitative risk management, and set thresholds
5. Leverage automation

The first point articulates the overall strategy of SDL: accountability for the security quality of software needs to reside primarily with the developers of the software. This creates incentives to make continuous improvements to security quality in the long term. Alternative accountability models, such as where the internal corporate security team takes responsibility for software security, don't scale well in our experience because of conflicting incentives between the business (release feature-rich software to customers) and risk management interests (ensure that security quality is high).

Of course, security assurance cannot be outsourced entirely to the development function, as that creates a "fox guarding the chicken coop" situation (i.e. lack of appropriate segregation of duties). So, point 2 notes that reviews conducted by (or overseen by) parties independent of the development team are necessary at key milestones. For example, the corporate security team could conduct pre-release penetration testing independently of the development team and track the remediation of identified issues.

Point 3 is perhaps self-evident, but nevertheless important: people have a hard time doing the right thing if they aren't told what's the right thing to do. Development team security training (including program managers, testers, and managers) is thus an important component of any SDL implementation. Training programs should provide job-relevant curricula, track comprehension (ideally linked to application on-the-job), and be supported by re-usable guidance, code libraries/routines, and checklists that developers can easily access on the job to enforce good behavior. Application security training could be the topic of an entirely separate discussion, so we'll say little else about it going forward other than to reiterate its importance to the success of the overall SDL effort.

Point 4 acknowledges that information security practices continue to evolve towards more mature, quantitative risk management approaches. These same principles are ideal to apply to software security assurance as well. For example, Microsoft's DREAD [12] risk rating system strives to quantify the severity of software vulnerabilities, and thus define the priority of remediation efforts. (DREAD is somewhat proprietary to Microsoft, but is illustrative of the concept of quantitative assessment; other risk quantification systems include CVSS2 [13], FAIR [14], and FMEA [15].) Beyond just the scoring system itself, it is important to establish thresholds for prioritization, or to put it colloquially, a "bug bar." The bug bar essentially defines for an organization the thresholds at which work will be done to remediate a flaw. It can be immensely helpful to define thoughtful thresholds like this in collaboration with all stakeholders in advance of performing assessments, to avoid disagreements over how to remediate flaws (resulting in delayed release, unacceptably risky flaws in released code, or both).

Point 5 needs little explanation. Automation yields greater efficiency, and SDL is no exception. Some key areas with high potential for improvement through automation include:

- Security code review (although the accu-

racy and relevance of output from common tools remains suspect)

- Fuzz testing (to be defined later)
- SDL process automation, e.g. self-help web portals for workflow management

We've included a brief overview of application security tools at the end of this article.

Pitfalls

We've covered some key SDL principles that can help improve the chance of good outcomes. Are there any practices that should be avoided?

Anecdotally, one of the main reasons for failure of an SDL initiative is lack of focus on benefits to the organization, and by extension, its customers. Many organizations approach SDL assuming that the implied virtues of more secure software will simply make it acceptable to all stakeholders. In addition, there is historical disagreement around whether focusing on return on investment (ROI) for security is worthwhile or achievable (we believe economic justification is imperative, and will return to this concept later). To counteract this tendency, we recommend developing a good "scouting report" on all stakeholders (especially customers!), how they perceive SDL, their key objectives, and expected performance indicators. Often, this basic research can point to simple and easily implemented initial steps that result in easy wins, good momentum, and a strong head start towards a sustainable SDL program.

Culture shock can also torpedo SDL implementations. The culture of software development generally resists structure and discipline, and specific group dynamics can present even further challenges. Often, security is perceived from the start as an outsider, and the various behavioral changes proposed within the SDL initiative are thus viewed with suspicion at the outset. Be prepared to adapt your specific SDL implementation to the general and specific culture of development within your organization to bypass culture shock and outsider perception out of the gate.

Those chartered with initiating SDL are often tempted to set unrealistic expectations for SDL outcomes in order to address the outsider perception issue. Obviously, this is not recommended. Software development cultures are often focused tightly on schedule and resource allocation, and individuals who mismanage those two fundamentals often sacrifice substantial reputational capital that is very difficult to re-acquire for subsequent release cycles.

Lack of alignment with other security initiatives can also introduce "audit fatigue" amongst developers, who typically bristle at being interrupted multiple times for what they perceive is the same issue. One of the typical examples here is web development shops that have to comply with PCI DSS. [16] They are faced with complying with both SDL and PCI-related security initiatives separately if those programs are not well-coordinated.

Finally, and perhaps most importantly, organizational governance is often neglected in the design of SDL programs. The common wisdom is to "get executive buy-in" for initiatives of this nature, and this is of course important. However, development cultures are more often driven by "bottom-up" perceptions, so it's important to consider lobbying of all stakeholders early and often.

SDL Implementation Examples

What does an SDL implementation look like in practice? As we've proposed, it should be well-aligned with the existing development rhythm and culture. Figure 2 shows a mock development lifecycle for a large enterprise.

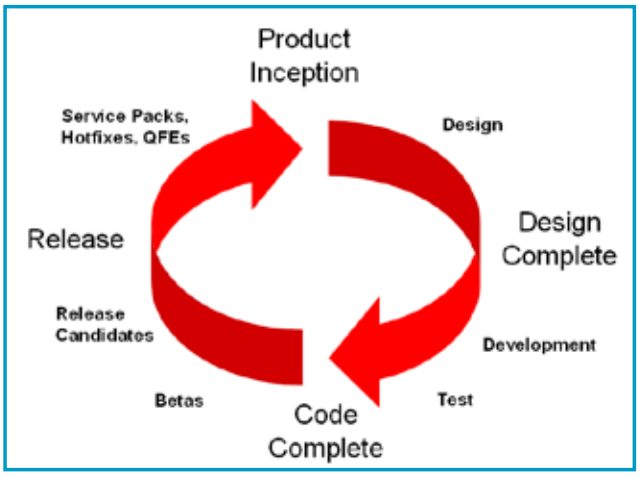


Figure 2: A mock large enterprise development lifecycle

Beginning with the lifecycle in Figure 2 as a baseline, in Figure 3 we overlay some common SDL milestones. This is one possible implementation for a large-scale organization with substantial resources.

It's important to note how Figure 3 aligns with the SDL principles we articulated earlier:

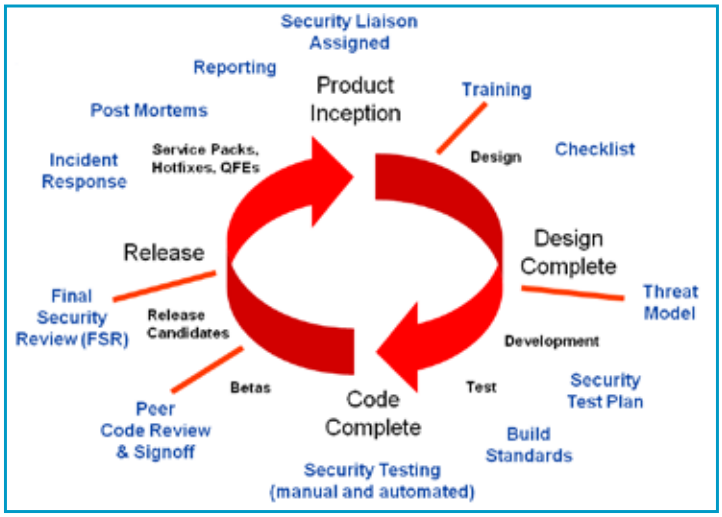


Figure 3: A sample SDL implementation overlaid on top of the previously introduced mock development cycle.

Principle	Implementation in Figure 3
Distribute the work of assessment and remediation, especially to the development team	A Security Liaison is assigned at project inception, who will be accountable for managing security workflow throughout. [2]
Independent review at key milestones	The red lines indicate milestones where independent review can occur. Note that these are closely aligned to existing development process gates.
Provide relevant training and re-usable guidance (checklists)	Training is an SDL gate that occurs early in the cycle. [3] Also, the "Build Standards" gate at the "Test" milestone illustrates an opportunity to provide re-usable checklists.
Strive for quantitative risk management, and set thresholds	The iterative nature of the overlay cycle provides multiple opportunities to check metrics (such as DREAD score mitigation) during the current and in future releases.
Leverage automation	A number of gates could require automated checks, such as the "Security Testing" and code review milestones.

2 Note that the security liaison manages workflow, not security outcomes, such as code security quality and other metrics. Ultimately, the development team leadership/executives are accountable for outcomes.

3 In practice, the number of developers who receive required training fluctuates between and within cycles, but the idea here is to enforce training early in a given cycle to ensure people have the training they require to do their jobs.

Although Figures 2 and 3 are illustrative of how SDL principles might be implemented on a large scale, we've stressed the importance of starting small with SDL and iteratively growing the program to a scale that is sustainable for a given organization. Figure 4 provides an example of a smaller-scale SDL implementation based on what we assert are the minimum components for success.

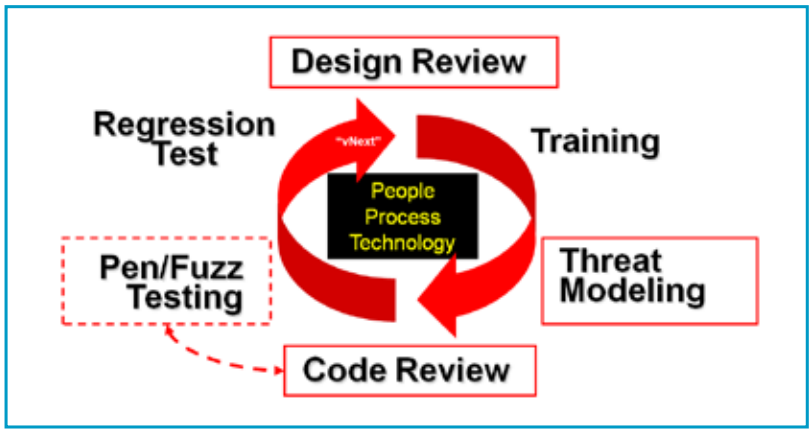


Figure 4: A light-weight SDL implementation example.

Note that many of the enterprise-scale SDL checkpoints (shown in Figure 3) have been eliminated in the example shown in Figure 4. The boxes highlighted in red in Figure 4 comprise an even more minimal SDL implementation made up of Design Review, Threat Modeling, and Code Review (with optional Penetration/Fuzz Testing). The terms shown in Figure 4 are defined in the Glossary at the end of this article.

Development Infrastructure

It's worthwhile to pause for a moment to highlight the importance of basic fundamental software development hygiene. Of course, many opinions exist (especially within development communities) about the exact meaning of "basic software development fundamentals," and we're not interested in starting such a debate here. Our primary point is that much of the theory and practice of SDL depend upon certain fundamentals being in place, and most SDL initiatives will not be successful without at least some structure to which it can be anchored. Some of the common key enablers of successful SDL implementations are listed in Table 1.

What About Alternative Programming Models?

The topic of software development fundamentals raises a popular question: Can SDL be applied successfully to iterative/unstructured Agile programming methods like Scrum and Extreme Programming? Absolutely, [17] but there is a point at which a lack of structure or too much "adaptive-ness" can hamper SDL. SDL is based on the premise that a minimal level of structure exists to which systematic evaluations of security quality can be anchored. If there is no structure to which anything can be anchored, then SDL will likely be challenging to implement. This again highlights the challenge of cultural integration for those without substantial development experience (e.g. security professionals attempting to implement SDL): it can be hard to differentiate natural resistance to change from active attempts to cover up an overall poorly managed existing development effort.

SDL ROI

To this point, we've discussed the "what" and some of the "how" of SDL. Before concluding, we'll take a step back and briefly survey the "why." Our basic premise is that, at least in business scenarios, the key driver of SDL should be economics, with emphasis on the "should be". Finding and interpreting data to support this contention is challenging.

Generally, there is a lack of systemic empirical evidence supporting measurable economic outcomes following implementation of SDL. Most studies to date have focused on hard operation costs yielding intangible benefits. For example:

- Savings-to-cost ratios ranging from break-even to 8 or 9 times [18]
- Average loss of 0.76% market value when a vulnerability is disclosed [19]
- Return on investment equated to 21% [20]

Other studies have shown the value of good design, user education, and automated response technology over finding and fixing bugs. [21, 21]

Microsoft has published data showing a sharp reduction in security bulletins published from Windows 2000 Server to Windows Server 2003. [23] Combining this data with separate claims by Microsoft that a security bulletin costs the company approximately $100,000 (in 2002 dollars) [24], one can impute that Microsoft saved approximately $3.7M due to their "Secure Windows Initiative" push that was one of the primary progenitors of their brand of SDL. Of course, this does not quantify tangible investment in SDL beforehand (let alone even as a percentage of overall spend); it just illustrates benefits in terms of hypothetically realized savings from un-issued bulletins.

Admittedly, this brief review of economic data in support of SDL has not done justice to the topic. Our sense based on anecdotal experience is that, like most things, the ideal risk/reward balance is not one-size-fits-all, but is rather best gleaned from experimentation and keen focus on tying SDL metrics to economic outcomes early and often. To end with one final piece of guidance on quantitative data supporting SDL, we paraphrase the Pareto Principle: Invest more in finding the "vital few" issues that cause the vast bulk of security vulnerabilities. We've provided one last table to help illustrate this point:

	Budget	Quantity Found	Quality Impact
React • Find & fix bugs • Bolt-on	20%	80%	20%
Prevent • Improve dev practices • Baked-in	80%	20%	80%

Table 2: Planning for SDL should focus on finding the "vital few" issues that cause the preponderance of security vulnerabilities.

Fundamental Dev Practice	Assists SDL By
Consistent test, build environments	Separating test and production data, ensuring expected run-time parameters
Concurrent versions system (CVS)	Enforcing known-state versioning and providing reversion capability
Defect management system	Provides a central repository for managing and measuring defect reduction
Reporting	Provides consistent communication of data to appropriate decision-makers

Table 1: Development practice fundamentals that enable SDL.

Joel Scambray, CISSP, is co-founder and CEO of Consciere, provider of strategic security advisory services. He has assisted organizations ranging from small startups to Microsoft Corp. address information security challenges and opportunities for over a dozen years, in diverse roles including consultant, corporate leader, entrepreneur, and co-author of the Hacking Exposed book series.
The author wishes to acknowledge important contributions to this article from Andre Gironda, Birgit Lahti, and Kevin Rich.

© Pitopia / Klaus-Peter Adler, 2007

How to conduct basic information security audits?

by Nadica Hrgarek

Managing information security has become more complex and the numbers of internal and external security threats are increasing. Conducting an information security audit makes good business sense to assess threats and security risks to information systems, to develop risk mitigation strategies and to ensure that identified security risks remain within acceptable levels. The purpose of this article is to provide guidance on how to conduct basic information security audits as part of the internal audit programme.

Introduction

Every business relies on the continuous and safe operation of information technology to maximize effective achievement of organizational goals. Today's organizations face a number of challenges in managing information security. Software vulnerabilities, unauthorized intrusions, increased spam delivery, infection of systems and data by viruses, worms or Trojan horses (backdoors), phishing attacks, denial of service (DoS) attacks, computer-assisted fraud, website defacements, economic espionage, laptop or mobile hardware theft, insider threats, non-compliance (e.g. security policy not followed by employees) and other threats to information security are just a few of many challenges for organizations. It is very important to note that information security is a continuous management process that requires ongoing attention and a well established risk management process.

To protect valuable IT assets such as computers and laptops, servers, networks, and sensitive data and to detect security risks/threats, companies should conduct regular information security audits. However, many companies ignore this important fact and do not perform their own basic information security audits for various reasons, such as lack of qualified auditors, lack of information security awareness and communication by top management, information security requirements and procedures are not established, costs to involve/hire IT security specialists being too high, etc. Another problem is that most internal auditors and IT employees are not familiar with information security audits.

Organizations that have implemented a quality management system in conformance with international standards (e.g. ISO 9001, ISO 13485, etc) can take advantage from internal audits when performing basic information security audits. The international quality management standards ISO 9001:2008 and ISO 13485:2003 require conducting internal audits (clause 8.2.2).

Internal audits should be conducted at defined intervals by independent and skilled internal auditors for management review and internal purposes. All elements of a quality management system (e.g. design control, process and production controls, corrective and preventive actions, software, management review, document and record control, etc) should be audited. The results of the internal audit activity should lead to continuous improvements, process innovations, risk mitigation, corrective and/or preventive actions, corrective action closures, lessons learned, observations, etc.

ISMS

ISO/IEC 27001:2005 is an information security management framework which helps organizations to establish a documented information security management system (ISMS). The purpose of ISMS is to ensure adequate and appropriate security controls that adequately protect information assets. ISO/IEC 27001:2005 standard can be used to assess compliance of the ISMS by internal and external parties. Clause 6 of this standard requires performing internal ISMS audits at planned intervals (at least once a year) to determine whether security objectives, controls, processes and procedures of the ISMS are compliant, efficient and executed as planned.

Internal ISMS audits shall be conducted by independent, competent and adequately trained auditors. Internal information security auditors should have the following knowledge and skills: communication and report writing skills, sufficient knowledge needed to perform technical security testing (i.e. knowledge in utilizing the penetration tools selected to detect vulnerabilities), working knowledge of application programming, network engineering skills, system administration skills, risk management skills, knowledge to interpret the requirements of information security standards in the context of an ISMS audit, skills to conduct an internal audit in accordance with ISO 19011:2002, etc.

The three core elements of information security (also known as the CIA triad) according to ISO/IEC 27001:2005 are:

- **Confidentiality** of information – protecting access to it from unauthorized users or systems.
- **Integrity** of information – safeguarding the accuracy and completeness of information and processing methods. Examples of integrity incidents are: an unauthorized user deletes a record from a database, changes a statement in a source code, executes an application, etc.
- **Availability** of information – ensuring information and associated assets are available to authorized users or systems when requested or needed. Examples of common availability incidents are: autho-

rized users are not able to access a website because of a DoS attack (e.g. ping of death, SYN flooding), loss of data processing capabilities as a result of natural disasters (e.g. fire, flood, earthquake, hurricane, etc) or human actions (e.g. bombs or strikes).

ISO/IEC 27002:2005 standard identifies 11 domain areas, 39 control objectives and 133 best practice security controls which help organize security policy and procedures. The eleven domain areas of ISO/IEC 27002:2005 are: (1) information security policy, (2) organizational security, (3) asset classification, (4) personnel/human resources security, (5) physical and environmental security, (6) communications and operations management, (7) access control, (8) information systems acquisition, development and maintenance, (9) information security incident management, (10) business continuity management, and (11) compliance with legal requirements.

Information security audit process

Security of information, computer systems, networks and the people who manage them are the focus of the information security audits. An information security audit is usually performed by competent auditors through interviews, vulnerability assessments, examinations of OS settings, analyses of network shares and historical data to identify all possible security risks. The information security audit should be built on past audit results to help refine the security policy and correct deficiencies which are discovered through the audit process. Table 2 lists some sample key questions that any information security audit should address to protect information assets against manipulation and destruction, to preserve availability, confidentiality and integrity of information.

Penetration tests should be complemented with information security audits and other security measures. These tests simulate an attack on an organization's systems and network; check if there are any improperly configured systems or other vulnerable systems. Penetration testing and other security tools (Table 3) allow auditors to discover the vulnerabilities. However, it is important to emphasize that some penetration testing tools can reproduce real attacks, which could cause systems crashing and compromise computer systems or network if they are not protected. Therefore, before conducting any penetration test on systems or networks, it is recommended to have consent from management. If internal resources are being used to perform penetration tests, those resources shall be independent and experienced penetration testers.

The following websites provide a review and brief description of currently available vulnerability assessment tools: www.securitywizardry.com, sectools.org.

Company assets to consider when conducting information security audits may include, but are not limited to:

- information assets (data files, user manuals, product specifications, procedures and work instructions),
- paper documents (contracts, procedures, guidelines),
- software assets (application software, system software, source code),
- hardware assets (computers, laptops, web servers, routers and networking equipment, printers, company smartphones/PDAs, scanners),
- personnel assets (users, administrators, developers),
- service assets,
- location assets.

The internal information security audit usually contains the following audit activities:

- *Planning and preparing the audit* – Appointing the audit team leader; defining audit objectives, scope and criteria; selecting an appropriate audit team; conducting a document review prior to the on-site audit activities (e.g. security policy and other applicable procedures, standards); preparing the audit plan; preparing work documents (e.g. audit checklist, audit sampling plans, forms for recording information); selecting appropriate audit tools and environment.
- *Conducting the on-site audit* – Conducting the opening meeting; review of documentation (e.g. hardware/software inventory, network architecture, incident logs, user account policy, password policy, backup policy, audit trails, etc); conducting interviews; review of location and environment controls (e.g. server room, fire extinguisher, fire protection door, etc); conducting technical assessment (e.g. running static and dynamic tools, firewall testing, checking system logs, checking system against known vulnerabilities, searching for privileged programs, checking all configuration files of running processes, checking extra network services, code review of non-standard programs, using social engineering techniques, etc); generating the audit findings; preparing the audit conclusions; conducting the closing meeting.
- *Generating, approving and distributing the audit report.*
- *Conducting audit follow-up activities* – Verification of completion and effectiveness of corrective/preventive actions arising from the internal audit.

An audit report should provide a complete, accurate and concise record of the audit. It should be prepared by the lead auditor and signed by the audit team members. Audit reports increase top management awareness of security issues and assist top management in decision-making processes. An audit report usually includes the following information:

- Audit reference number,
- Date of audit,
- Identification of lead auditor and audit team members,
- Executive summary,
- Audit criteria,
- Audit findings (e.g. non-conformities, observations),
- Recommendations for the audit findings (i.e. corrective, preventive or improvement actions),
- Audit conclusions,
- Appendices.

Conclusion

There are many benefits in performing information security audits: ensuring business continuity, minimizing business damage (e.g. preventing financial and availability losses, avoiding image loss), improved enterprise security, better risk management process, gaining deeper knowledge of different aspects of security, measuring compliance with current security policies and procedures, etc. Security audits should be performed at regular intervals or after any significant infrastructure or software changes.

Due to the global economic downturn, IT security budgets are tight and top management needs to understand how information security audits help to detect security threats, improve safeguarding of assets and ultimately decrease costs. Top management support and commitment to the information security is one of the key success factors in any effective information security project.

To obtain perfect security is not possible, and therefore the costs of information security should be commensurate with the business needs and security risks of any computer system.

Number	Title
ISO/IEC 13335-1:2004	Information technology – Security techniques – Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management
ISO/IEC 18028-1:2006	Information technology – Security techniques – IT network security – Part 1: Network security management
ISO/IEC 18028-2:2006	Information technology – Security techniques – IT network security – Part 2: Network security management
ISO/IEC 18028-3:2005	Information technology – Security techniques – IT network security – Part 3: Securing communications between networks using security gateways
ISO/IEC 18028-4:2005	Information technology – Security techniques – IT network security – Part 4: Securing remote access
ISO/IEC 18028-5:2006	Information technology – Security techniques – IT network security – Part 5: Securing communications across networks using virtual private networks
ISO/IEC 18045:2008	Information technology – Security techniques – Methodology for IT security evaluation
ISO/IEC 27000:2009 *under development*	Information technology – Security techniques – Information security management systems – Overview and vocabulary
ISO/IEC 27001:2005	Information technology – Security techniques – Information security management systems – Requirements
ISO/IEC 27002:2005	Information technology – Security techniques – Code of practice for information security management
ISO/IEC FCD 27003 *under development*	nformation technology – Security techniques – Information security management system implementation guidance
ISO/IEC FCD 27004.2 *under development*	Information technology – Security techniques – Information security management – Measurement
ISO/IEC 27005:2008	Information technology – Security techniques – Information security risk management
ISO/IEC 27006:2007	Information technology – Security techniques – Requirements for bodies providing audit and certification of information security management systems
ISO/IEC WD 27007 *under development*	Information technology – Security techniques – Guidelines for information security management systems auditing

Table 1: Overview of common ISO/IEC standards for IT/information security

Security Area	Sample audit question
Logical security	• Have all custom-developed applications been written with security in mind? • Have custom-developed applications been tested for security flaws?
	• Have the operating systems and commercial applications been updated with the appropriate security patches? • Have the security patches been tested before deployment?
User identification and authentication	• Are user names (IDs) unique and linked to real persons? • Is there a defined number of consecutive unsuccessful attempts to login?
User password management	• Are passwords in place? (e.g. minimum password length, password expiration, password reusability, disabled passwords that are no longer valid) • Are the users informed and asked to follow good security practices in selection and use of passwords? • Are all passwords changed regularly, especially the system administrator's? • How difficult are passwords to crack?
Virus protection	• Is anti-virus software installed on all computers? • Are personal computers regularly scanned for malware (e.g. computer viruses, logic bombs, spyware/adware)? • Is a procedure for automatically updating the anti-virus software in place?
Access control	• Are there access control lists (ACLs) in place for network assets to control who has access to shared data? • Are there access control lists in place for applications and information? • Are there access control lists in place for mobile computing and teleworking? • Are the user access rights reviewed at regular intervals and revised, if necessary? • Are there audit logs to record who has accessed data? • Are the audit logs reviewed?
Data backup and recovery	• How is backup media stored? • Who has access to backup media? • Is backup media up-to-date? • At what frequency are backups taken and tested? • Is there a method for performing a restore of the data?
Configuration and application change management	• How are configurations and code changes documented? • Do all system changes go through a formal change control process? • Have changes been tested before being placed into production? • How are records reviewed? • Who conducts the reviews?

Security Area	Sample audit question
User support	• Is user documentation (user manuals, online help, etc) available and up-to-date? • Have users been trained in the proper use and security of applications they use? • Is there a process for user improvement requests?
Disaster recovery	• Is there a disaster recovery plan in place? • Has a disaster recovery plan been reviewed and approved? • Are disaster recovery teams established to support disaster recovery plan? • Are recovery plans regularly tested? • Is there at least one copy of company's data and application software stored in a secure, off-site location? • Does a hardware maintenance contract exist with a supplier?
Information security policy	• Is there a documented security policy in place? • Who is the owner of the information security policy? • Who is responsible for its review according to a defined review process? • Has security policy been communicated to all employees and contractors? • Is there a clear-screen policy in place? • Is there a clear-desk policy to ensure that employees secure confidential files when they are not working on them?
Security awareness and training	• Is there security awareness and is an appropriate information security training program in place? • Have all copies of software been properly licensed and registered?
Physical and environmental security	• Are systems left logged in while employees are away? • Has physical protection against external and environmental threats (e.g. fire, flood, earthquake, explosion, electromagnetic interference) been designed and applied? • Has physical security for offices, rooms and facilities been designed and applied? • Are there physical entry controls to protect secure areas and to ensure that only authorised personnel are allowed access? • Are there physical protection and guidelines for working in secure areas? • Has equipment been protected from power failures and failures caused by other utilities? • Has equipment been correctly maintained to ensure availability and integrity? • Are network servers physically secure in a separate area?

Table 2: Sample generic information security audit checklist

Name of Tool	URL	Description
Nessus®	www.nessus.org	Remote network vulnerability scanner
SAINT®	www.saintcorporation.com	Network vulnerability scanner
IBM® Internet Scanner	www.iss.net	Network vulnerability scanner
Retina®	www.eeye.com	Network security scanner
QualysGuard® Suite	www.qualys.com	Tools for vulnerability management, policy compliance, PCI (Payment Card Industry) compliance, and web application scanning
CORE IMPACT Pro	www.coresecurity.com	Automated penetration security testing software
SATAN	www.porcupine.org/satan	Security administrator tool for analyzing networks
Nmap (Network Mapper)	nmap.org	Free and open source utility for network exploration and security auditing (port scanner)
John the Ripper	www.openwall.com/john	Multi-platform password hash cracker for detection of weak passwords
Crack	ftp.cerias.purdue.edu/pub/tools/unix/pwdutils/crack/	Password cracker
Tiger	www.nongnu.org/tiger	Unix security audit and intrusion detection tool
COPS (Computer Oracle and Password System)	www.nongnu.org/tiger ftp.cerias.purdue.edu/pub/tools/unix/scanners/cops/	System monitoring tool
Foundstone	www.foundstone.com	Many free tools and resources: forensic tools, Foundstone SASS (Software Application Security Services) tools, intrusion detection tools, scanning tools, stress testing tools, etc

Table 3: Overview of commonly used information security audit tools

Basic information security terms

An **asset** is anything that has value to the organization. [ISO/IEC 13335-1:2004] Assets are subject to many kinds of threats.

A **threat** is a potential cause of an unwanted incident, which may result in harm to a system or organization. [ISO/IEC 27001:2005] An example of a threat could be the accidental deletion of system data.

Vulnerability is defined as a weakness of an asset or group of assets that can be exploited by one or more threats. [After ISO/IEC 27001:2005] Vulnerabilities can be found in software, information systems, network protocols and devices, etc. If vulnerability is not managed, it will allow a threat to materialize. Examples of vulnerability are: unpatched software, weak passwords, lack of access control, no firewall installed, insufficient security training, unlocked doors and windows, shared accounts, programming input validation errors, etc.

A **risk** is the potential that a given threat will exploit vulnerabilities to cause loss or damage to an asset or group of information assets and thereby cause harm to the organization. It is measured in terms of a combination of the probability of an event and the severity of its consequences. [After ISO/IEC 13335-1:2004]

Information is an asset which, like other important business assets, has value to an organization and consequently needs to be suitably protected. [ISO/IEC 27002:2005] Information can be in various forms: printed or written on paper, stored electronically, transmitted by post or e-mail, shown on corporate videos, spoken in conversation or exists as knowledge acquired by individuals.

Information security is preservation of confidentiality, integrity and availability of information; in addition, other properties, such as authenticity, accountability, non-repudiation, and reliability can also be involved. [ISO 27002:2005]

Industrial espionage is unauthorized collection of confidential, classified or proprietary documents.

Nadica Hrgarek holds a B.Sc. in information systems and a Master of Science in information science from the University of Zagreb, Croatia.
Since March 2007 she has been a member of the RA/QA department at MED-EL Elektromedizinische Geräte GmbH (www.medel.com), a hearing implant company located in Innsbruck, Austria. Nadica is currently working as a Senior QA Specialist – Quality Improvement. Her responsibility covers all aspects of quality improvements ranging from coordination of corrective and preventive actions, non-product software validation support, conducting training, internal and supplier audits, etc.
Nadica is a certified ISTQB tester (full advanced level), ISO 9000 internal and lead auditor and has conducted more than 20 internal, product, process and supplier audits.
She is co-founder and Head of the Advisory Board of the Croatian Testing Board (CTB), which was founded in 2008. She is also a member of the German Association for Software Quality and Training (ASQF).

Masthead

EDITOR
Díaz & Hilterscheid
Unternehmensberatung GmbH
Kurfürstendamm 179
10707 Berlin, Germany

Phone: +49 (0)30 74 76 28-0 Fax: +49 (0)30 74 76 28-99 E-Mail: info@diazhilterscheid.de

Díaz & Hilterscheid is a member of "Verband der Zeitschriftenverleger Berlin-Brandenburg e.V."

EDITORIAL
José Díaz

CHIEF ADVISOR
Stephan Goericke

LAYOUT & DESIGN
Katrin Schülke

WEBSITE
www.securityacts.com

ARTICLES & AUTHORS
editorial@securityacts.com

ADVERTISEMENTS
sales@securityacts.com

PRICE
online version: free of charge

EDITORIAL BOARD
Prof. Dr. Sachar Paulus
Manu Cohen-Yashar
Markus Schumacher
Aaron Cohen

In all publications Díaz & Hilterscheid Unternehmensberatung GmbH makes every effort to respect the copyright of graphics and texts used, to make use of its own graphics and texts and to utilise public domain graphics and texts.

All brands and trademarks mentioned, where applicable, registered by third-parties are subject without restriction to the provisions of ruling labelling legislation and the rights of ownership of the registered owners. The mere mention of a trademark in no way allows the conclusion to be drawn that it is not protected by the rights of third parties.

The copyright for published material created by Díaz & Hilterscheid Unternehmensberatung GmbH remains the author's property. The duplication or use of such graphics or texts in other electronic or printed media is not permitted without the express consent of Díaz & Hilterscheid Unternehmensberatung GmbH.

The opinions expressed within the articles and contents herein do not necessarily express those of the publisher. Only the authors are responsible for the content of their articles.

No material in this publication may be reproduced in any form without permission. Reprints of individual articles available.

Index Of Advertisers

Training with a View

Díaz Hilterscheid

also onsite training worldwide in German, English, Spanish, French at
http://training.diazhilterscheid.com/
training@diazhilterscheid.com

"A casual lecture style by Mr. Lieblang, and dry, incisive comments in-between. My attention was correspondingly high. With this preparation the exam was easy."
Mirko Gossler, T-Systems Multimedia Solutions GmbH

"Thanks for the entertaining introduction to a complex topic and the thorough preparation for the certification. Who would have thought that ravens and cockroaches can be so important in software testing"
Gerlinde Suling, Siemens AG

- subject to modifications -

09.11.09-11.11.09	Certified Tester Foundation Level	Berlin
30.11.09-04.12.09	Certified Tester Advanced Level - TESTMANAGER	Berlin
02.12.09-04.12.09	ISSECO® - Certified Professional for Secure Software Engineering	Frankfurt a. M./Neu Isenburg
07.12.09-10.12.09	Certified Tester Foundation Level	Frankfurt am Main
14.12.09-17.12.09	Certified Tester Foundation Level	Düsseldorf/Köln
14.12.09-18.12.09	Certified Tester Advanced Level - TEST ANALYST	Berlin
05.01.10-07.01.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
18.01.10-20.01.10	Certified Tester Foundation Level - Kompaktkurs	Stuttgart
18.01.10-22.01.10	Certified Tester Advanced Level - TEST ANALYST	Frankfurt am Mein
25.01.10-29.01.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
08.02.10-10.02.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
15.02.10-19.02.10	Certified Tester - TECHNICAL TEST ANALYST	Berlin
22.02.10-25.02.10	Certified Tester Foundation Level	Frankfurt am Man
22.02.10-26.02.10	Certified Tester Advanced Level - TESTMANAGER	Düsseldorf
24.02.10-26.02.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
01.03.10-03.03.10	ISSECO® - Certified Professional for Secure Software Engineering	Berlin
08.03.10-10.03.10	Certified Tester Foundation Level - Kompaktkurs	München
15.03.10-17.03.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
15.03.10-19.03.10	Certified Tester Advanced Level - TEST ANALYST	Düsseldorf
22.03.10-26.03.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
12.04.10-15.04.10	Certified Tester Foundation Level	Berlin
19.04.10-21.04.10	Certified Tester Foundation Level - Kompaktkurs	Hamburg
21.04.10-23.04.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
28.04.10-30.04.10	Certified Tester Foundation Level - Kompaktkurs	Düsseldorf
03.05.10-07.05.10	Certified Tester Advanced Level - TESTMANAGER	Frankfurt am Main
03.05.10-07.05.10	Certified Tester - TECHNICAL TEST ANALYST	Berlin
10.05.10-12.05.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
17.05.10-21.05.10	Certified Tester Advanced Level - TEST ANALYST	Berlin
07.06.10-09.06.10	Certified Tester Foundation Level - Kompaktkurs	Hannover
09.06.10-11.06.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
14.06.10-18.06.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
21.06.10-24.06.10	Certified Tester Foundation Level	Dresden

Kurfürstendamm, Berlin © Katrin Schülke