Guidelines to develop secure applications

Alberto Partida
GSEC Version 1.4b Option 1
March 7, 2003

Abstract

This paper provides some security guidelines to application development teams, especially those developing web applications. These guidelines do not substitute currently accepted project (and application) development methodologies but attempt to serve as a brief security reference and as an invitation to consult and, eventually, follow the references referred on this paper.

These guidelines aim to achieve that a project and/or application plan and budget reflect the complexity of taking security measures into account in a pre-emptive manner.

The following sections present security principles to be incorporated in all the phases an application goes through, from the moment a business area in an organisation comes up with an idea to implement a business function, until an application is launched into production.

Special attention is paid on how to reach specific levels of security objectives required by applications. Additionally, topics such as firewall-compliant applications and secure session maintenance are briefly discussed in these pages.

Introduction

Implementing security measures throughout the whole application lifecycle is definitively more cost and security-effective than adding some security mechanisms at the very end of the development phase. Just before launching an application into production, when time deadlines are tight, development teams are not especially receptive to new and unexpected security recommendations due to time pressure.

Normally, applications are developed within the scope of a specific project. Therefore, it is important to link the following security guidelines with the phases of a project methodology. Aiming to be generic, this paper will not focus on a specific project and application methodology but it will associate the proposed security measures with typical phases of a project.

Business functions

Every application plays a business function within an organisation. The objective of an application is playing this business function with a predetermined level of quality and security. These levels are evaluated through application controls. Among these controls, security controls are a subset of them.

All security controls exposed in this paper aim to achieve the following security objectives. [Maung] also refers to security objectives (especially the three first ones) when explaining how an application should prepare for a web security review:

a. Confidentiality: The property that avoids information to be revealed or available to non-authorised parties.
b. Integrity: The property that avoids information to be modified by non-authorised parties.
c. Availability: The property that allows information to be accessed and used by authorised parties.
d. Authentication: The property that guarantees that the identification has been accomplished with enough truthfulness.
e. Authorisation: The property that guarantees that a user (either a process or an individual) is allowed to access a specific function.
f. Auditability: The property that allows the elaboration of a detailed log of all events related to an application.
g. Non-repudiation: The property that guarantees that some specific data comes from a determined user (as mentioned in [Peteanu, page 38], there are at least eight types of non-repudiation attending to the different actions to be non-repudiated).

These are functional definitions (mostly based on experience) of the security objectives. Alternative definitions of these concepts can be found in [Krutz, pages 3-4].

Security questionnaire

Not all applications in an organisation require the same level of security. Applying the terms introduced in the previous section, every application requires a specific level in each security objective, according to the business function it plays.

Therefore, it is proposed to elaborate a brief security questionnaire so that the business area can provide the application development team (or, in a broader scope, the project team) with a clear picture of what the application require from the security standpoint. In this paper, the group providing application security advice is considered to be part of the application development group.

This questionnaire will clarify the optimal level required by the future application in every security objective. The following questions are a sample of what could be inserted in the questionnaire:

- Does the business function deal with confidential information?
- Does the business function require to authenticate users?
- Does the business function require integrity controls?

As these questions are to be answered by business-oriented teams, it is recommendable to translate them into 'business terms' such as:

- How will affect the organisation this business information made public?
- Does the business function require to distinguish between users?
- What would happen if the information required by the business function is modified by someone?

Requirement analysis

Business requirements: All business requirements will be established and accepted before the specification and the detailed design of the application is started. Security requirements are a substantial part of the business requirements [Cresson, measure 166]. Business requirements will be identified following a structural approach and will consider:

- All information processed by the system (inputs and outputs).
- Contractual restrictions and obligations.
- Business area objectives and needs.

Security requirements

To explain the link between risk and security requirements, three definitions extracted from [Krutz, pages 16-20] are provided:

- Threat: The occurrence of any event that causes an undesirable impact on the organisation (i.e. any potential danger).
- Vulnerability: The absence or weakness of a safeguard.
- Risk: The likelihood of a threat taking advantage of a vulnerability.

Once the development team obtains a clear picture of the business function and its security needs, the elaboration of the security requirements basically consists of the translation of the business needs into specific technical security measures. These measures will keep the business risk at a reasonable and acceptable level.

This translation of business needs into security measures occurs after the business risk analysis, an essential initial phase of the project in which the possibility of something unplanned happening (i.e. a threat taking advantage of a vulnerability) is analysed. Once the business risk is known, requirements, priorities and processes are understood and agreed by the project team. There are plenty of different proposals to classify risks. For example, a risk matrix can be found in [Maung].

The following sections provide a series of security recommendations that could well be the body of a checklist that will help development teams to bear security in mind while developing a new application.

Application approach

A definer and a translator: The project team will count on two responsibility positions, one responsible for the clear definition of business requirements and the other responsible for the translation of these requirements into technical features.

Development team isolation: Development teams will be functionally isolated from production/operational teams. This ensures the application maintainability by groups that were not involved in its development.

Development tools: All software and hardware used to develop code will be selected from a list of secure software and hardware already accepted and tested by the organisation.

Formal description: All in-house developed software will have a formal specification in a written form with a twofold objective: acting as a description of the agreement reached between the business area and the development team and being a reference for future teams to maintain or update the application.

Modularity: All code developed in-house will be coded modularly so that it can be added to a global repository and afterwards used in future applications. Therefore, it is important to describe the functionality of every module in a standard mode using formal specifications. This is especially important in all security modules developed in-house [Cresson, measure 153].

Security modules: As soon as a collection of security services providing security objectives is available to applications in the form of security modules which have been implemented and tested, the provision of security services in a homogeneous way will be immediate. As an example of this, [Peteanu, page 46] describes the separation between the application and the provider of security services as the main goal of the use of the Generic Security Service Application Program Interface (GSS-API).

Architecture: Although this paper does not include architectural guidelines, from the security viewpoint it is important to recommend the creation of these four different architectural elements (mainly at the server side):

- Presentation logic: This layer will present information to the user, performing only this interface function. It will be detached from the rest of application components, since clients will normally interact with this interface.
- Business logic: An intermediate layer acting as a core gateway between the presentation logic and the information storage. This component accomplishes required business tasks.
- Information storage: An element, not directly in contact with the user, where all required and produced business information is stored.
- Security services: An isolated element where most security processes dealing with the different security objectives are performed (e.g. user authentication and authorisation).

Communication flows between these elements will be controlled and optimised. Some references, for example [Owasp, pages 13-14], propose similar architectural elements, although their names and functions could vary slightly from the ones presented here.

Documentation: Application documentation will be complete to facilitate the application's maintenance and operation by groups which were not involved in the development of the application [Cresson, measure 156]. Access to this documentation will be granted on a 'need-to-know' basis. Typical items of the documentation are:

- Design specification (mentioned in the 'formal description' subsection).
- Module descriptions.
- Dependencies with other software.
- Installation and operational procedures.

Development environment: All development environments, including hardware and software, will be physically and logically isolated from production environments. Only development teams will access software that is being developed. The introduction of uncontrolled and unchecked software will be avoided. This risk can be mitigated by the use of integrity checkers and version control software.

Outsourcing: When an external provider develops the application, a contractual framework stating all previous points will be in place [CRAMM group 110 - subgroup 220]. Additionally, the external provider will put in place security measures according to the nature of information required in the development. These measures will be identical to the ones followed internally by the customer organisation.

Application design

Once all security requirements are defined, the next step is introducing these security measures in the application's design. Apart from these specific security requirements, developers will keep in mind the following design principles. Some of them have been extracted from [Owasp, pages 10-11]:

Complexity avoidance: The design will be kept as simple and easy as possible. A small and simple piece of code is less prone to errors and easier to maintain than a complex code.

Use of standard protocols and technologies: Standard protocols and technologies are subject to public scrutiny and therefore, easier to maintain in an updated and secure state. Furthermore, there is no need to re-invent security mechanisms when some standard ones already exist as mentioned in [Peteanu, page 34] (e.g. SSL/TLS).

Code re-use: As already explained in the previous sections titled modularity and security modules, new lines of code will be saved when using already existing modules providing the desired functionality.

Open design: Security solutions used by applications will be based on public mechanisms. The robustness of a security mechanism will be based on the robustness of the used keys and algorithms (and not on the ’obscurity’ of the mechanisms used).

Complete mediation: Each access managed by the application will be controlled with an appropriate authorisation e.g. every access to every object will be checked before it is granted.

Least privilege: All users and processes will have a minimum set of privileges sufficient to play their planned tasks. This principle limits the damage caused in an application by an attacker misusing privileges given to a user or a process.

Deny by default: Access privileges will always be explicitly granted on demand and not given by default.

Separation of privileges: It is important to isolate privileges needed in an application for different tasks and in different periods of time. Consequently, an intruder taking control of a task will not be able to execute additional tasks.

Least common mechanism: Users will be isolated from each other in applications, so that a user’s actions will not affect others’. Therefore, shared resources will be kept to a minimum.

Mistrust the unknown: Information provided by users or external processes will not be trusted (e.g. a form filled by an unknown user in a front-end such as a web browser). Format, length and integrity checks will be performed on every data coming from an uncontrolled source. The main goal of this principle is avoiding 'semantic injection'. [Peteanu, page 50] uses this expressive term when ‘an application does not distinguish between data and language as input’. For example, these will be some checks:

- Compulsory data is actually provided by the user.
- Data length does not exceed the expected one.
- Data format is the valid one (e.g. numeric, text field, etc.).
- Data is between the expected range of values (e.g. a user’s date of birth will not be a date occurring in the future).
- Data is compatible with already inserted data.
- Data is sequential when required (e.g. a user is requested to insert her car's description only after she has answered that she owns a car).

[Brassine] also justifies why input data checks are extremely important especially in web applications.

Data consistency: Data files will be strictly controlled using e.g. integrity checkers based on hashing/digesting algorithms.

Error control: Avoid unexpected disruptions of the application so that it can recover from every unexpected event happening to the application. Identify the possible error causes (e.g. data input, file access or network access errors) and treat them separately.

Generic messages: When user interaction is required after an error has occurred, send generic or descriptive messages to the user. Error messages due to a failure in a security mechanism (e.g. either the username or the password not inserted correctly) will always trigger generic messages in order no to give useful hints to a possible attacker.

Safe failure: Whenever the application crashes, the underlying system will remain in a safe and stable state, without allowing other systems or resources to be accessed. This recommendation can be linked with the 'trusted path' concept described in [Peteanu, page 23], where it is proposed always to have a direct communication path between a user and the platform a user can trust (i.e. the Trusted Computer Base, TCB, defined in [TCSEC]).

Defence in depth: It is not wise to blindly rely on a unique security mechanism. There will probably be a time when unfortunately, it will be vulnerable and eventually exploited. Therefore, different and complementary security mechanisms will always be put in place, so that if one of them fails, the rest will still be effective (see [Owasp, page 10]).

More specific information about how to develop a secure web application can be found in [Owasp]. A comprehensive and easy-to-read study, recommendable to be considered before starting the design of an application.

Data confidentiality

Applications requiring data confidentiality will fulfil, up to a certain extent, the following security requirements. The number of requirements to be followed will depend on the level of confidentiality required to be achieved:

Transmitted information: Information in transit between the different components of the application (e.g. between the client and the server) will be protected. This is technically accomplished with the use of cryptography (e.g. the use SSL/TLS in web applications has become an 'industry standard')[CRAMM 140- 255].

Stored information: Mirroring the previous rationale, information in storage will be protected by the use of cryptography [CRAMM 20-65].

Authentication information: A typical example of information to be encrypted is user authentication credentials such as passwords, when they are both transmitted and stored, so that user impersonation is not feasible.

Cryptography: Depending on the sensitivity of the data, different cryptographic techniques at the application level will be applied. These techniques will be publicly scrutinised and accepted algorithms (e.g. symmetric ciphers, public-key schemes or even a simple transposition or substitution algorithm). The robustness of these solutions will always be based on the robustness of the keys and algorithms used (as already mentioned when describing the 'open design' principle).

Data integrity

Applications requiring data integrity will fulfil, up to a certain extent, the following security requirements. The number of requirements to be followed will depend on the required level of integrity. The requirements presented in this section follow the structure already presented in the data confidentiality section:

- Transmitted and stored information.
- Cryptographic methods to achieve this integrity checks.

Transmitted information: Information in transit will be integrity checked. This is technically possible using cryptography (e.g. the use SSL/TLS in web applications has become an ‘industry standard’)[CRAMM 180-320 and 20-95].

Stored information: Similarly, information in storage will be integrity checked by the use of cryptography.

Cryptography: Integrity checking algorithms used at application level will be publicly scrutinised and accepted algorithms (e.g. hash functions or even a simple message checksums). The robustness of these solutions will always be based on the robustness of the algorithms used (as already mentioned when describing the ‘open design’ proposal). Algorithms currently used are e.g. MD5 and SHA-1.

Information availability

The number of availability requirements to be followed by an application will depend on the level of availability required:

Backup policy: Application information (information produced or required by the application) will always be backed up to prevent information losses. Backup copies will reside in a different location and they will be periodically restored to ensure that they are fully available at any time.

The organisation’s backup policy will reflect business needs. Backup and restoring procedures will state all steps and frequencies to perform these tasks.

Redundancy: Sometimes only backup data is not enough to guarantee the business’ availability requirements. In these cases, a fully redundant application (or even network) platform will be required.

User and process authentication

The number of authentication requirements to be followed by an application will depend on the level of authentication required:

Credentials: Authentication is normally achieved by presenting authentication credentials to the system. These credentials can be based on:

- Something you know (e.g. the typical username/password pair).
- Something you have (e.g. a secure token, a chip card, a certificate).

- Something you are (e.g. your fingerprint is recognised by a biometric device).

Cryptography: Authentication credentials will be protected when they are transmitted and stored. In this case, protection means that credentials are not subject to be used by any other individual. As mentioned already in the ‘data confidentiality’ section, this is normally achieved using cryptography (e.g. passwords will be stored hashed to avoid an administrator impersonating a user).

Maintainability: A policy stating how to issue, use, renovate and discard credentials, as well as a list of requirements to be fulfilled by these credentials (e.g. a password policy stating their length, format and expiration period) will be elaborated.

[Owasp, pages 16-21] provides more specific information about authentication in HTTP-based applications.

Authenticating processes: Process credentials will never be inserted in a piece of code. This prevents sound maintenance of credentials and opens an easy way for an attacker to enter e.g. a database. As an alternative, these credentials will be stored in a controlled environment with only the minimum necessary access rights and, eventually, protected by cryptographic techniques and audit controls.

Secure session maintenance: This section is specially dedicated to HTTP-based applications. HTTP (and also HTTPS) is a stateless protocol. This means that the protocol by itself does not provide a way to maintain a session (i.e. user information transmitted from server to client and vice versa). Sessions will be maintained deploying session tokens. As [Owasp, page 23] states, these session tokens will be:

- User unique.
- Non-predictable.
- Non-persistent.
- Resistant to reverse engineering (i.e. it will be unfeasible to guess the information they convey).

There are three different ways to maintain a session:

- Hidden HTML fields in a web page.
- Static or dynamically rewritten URLs.
- Cookies (preferably using Javascript).

Additionally, the capture and fraudulent use of these session tokens through the network will be avoided. There are two possible measures to mitigate this risk:

- If the network used is not trusted (e.g. a public network) a cryptographic solution such as SSL/TLS avoids the interception of session tokens.

- Session tokens will be built on information from a specific HTTP client such as page tokens (as mentioned in [Owasp]), but never based on user specific information (username, password, etc.).
- Session tokens will expire.

User and process authorisation

The number of authorisation requirements to be followed by an application will depend on the level of authorisation required:

[Owasp] and [Krutz] mention different access control mechanisms. This paper will only focus on the Role Based Access Control Model (RBAC) since it is widely used in web applications. This model enables:

- Segregation of duties (an important concept in security, allowing complementary tasks to be done only by different entities).
- Easier maintainability of privileges and rights within an application.
- The creation of a general portal from which all corporate applications could be accessed.

User groups: According to the function they play, users will be included in user groups. A group could be a member of another groups (i.e. there will be nested groups).

Roles: A role is a specific pattern of use identified within an application according to the different business activities performed (i.e. some users can read information, some others can edit, and others can delete). According to business needs, roles will be assigned to the different existing groups (always following the ‘least necessary privilege’ principle).

The schema presented in this section fits very well with technologies such as LDAP directories.

Data auditability

Applications normally require the logging of occurred events for auditing, statistical and operational purposes among others. The number of requirements to be followed will depend on the required level of auditability in an application:

Configurable and exportable event log: Data to be logged will be configurable, a typical collection of events to be logged is the following:

- User identifier.
- Date and time of the event.
- Type of event.
- Accessed files and called modules.

As mentioned in [Peteanu, page 38], audit data formats will be easy to read and parse. There will be the possibility to export audit data (i.e. using a standard and open format).

Business event log: Events logging business events (according to business requirements) will be logged in a access-controlled environment.

System event log: Some system events will be logged regardless of specific business requirements. These logs will enable administrators to control and review the operation of the system. This is a detailed list of system events to be logged:

- Initialisation, boot-up and stop of the application.
- Failed access attempts.
- Privileged operations (i.e. those requiring especial access rights).
- Use of any powerful user account.
- Access rights updates.
- Configuration and security file updates.
- External modules invoked from the application.
- Access to sensitive information.

Non-repudiation of data

As mentioned already when describing the security objectives, [Peteanu, page 38] mentions that no less than eight types of non-repudiation can be distinguished, basically they refer to the different phases of a message-based communication:

- Approval of a message's content.
- Sending/submission of a message.
- Transport of a message by a delivery authority.
- Receipt and knowledge of a content.

The technical mechanism also used to achieve non-repudiation is cryptography (mainly hashing and signing algorithms). However, most applications still do not require highly sophisticated non-repudiation mechanisms (i.e. signing algorithms meet current requirements). If the business function requires so, legal advice will be sought, since the non-repudiation mechanism will have to provide enough guarantee to bind an action with a user.

Firewalling capabilities

Wherever your application is going to run, it is probable that information channels will have to cross through one or more application level firewalls (plus through several routing devices deploying access control lists).

Therefore, application protocols will follow these recommendations:

Standard protocols: Network protocols will be IP-based. Preferably TCP (connection-oriented protocol) to UDP (connectionless protocol). This will facilitate the filtering and the control of the information flow.

Unique channel: Communication sessions between different elements (normally between client and server) will use a limited and known number of ports (i.e. a unique channel is preferable to a set of channels).

Client role: It is important that the client always keep its role i.e. all communications will be initiated by the client and not by the server.

Network information in the payload: Protocols will avoid the sending of necessary network information (e.g. IP addresses) in the data payload. This ensures that typical networking procedures such as network address translation can be applied at any moment.

Proxy capabilities: It is desirable that application protocols cater for the possibility of setting up a proxy element between client and server (acting as a client towards the server and vice versa). At this point, some architects and developers will certainly think that this is easier said than done!

Software testing

Once the application has been developed, functional, stress and security testing will be performed. These tests will follow an agreed and accepted methodology so that test cases and, afterwards, test results, can be fully scrutinised and approved by the system owner.

Production data: Production data (especially sensitive, critical, private and personal data) will be avoided in all testing environments [see Cresson, measure 146]. If the use of production data is unavoidable, then:

- This data will be accessible to developers only during the testing phase.
- Security measures in the testing environment will be identical to the measures present in production environments.

Testing group; Application testing groups will be different to the development group to guarantee a total distinction between developing and testing tasks.

Test plans: All test plans will have a clear script. A proposal will be:

- Pre-requirements to perform the test.
- Objective of the test.
- Necessary data for the test.
- Steps to perform the test.
- Expected test results.

Security tests: As stated in [CRAMM 60-160], all security requirements will be tested to guarantee that agreed security measures have been put in place. Security test plans will comprise all steps to check all security put in place in the application. A comprehensive proposal of a security testing methodology can be found in [Herzog].

Production environment

Isolated installation: After the application has been fully tested and a production environment (isolated from development) has been put in place, the system will be installed and launched into production.

No development tools: Production environments will not have development tools such as compilers [CRAMM 240-385]. Source and object files will not reside in the production system.

Change management: Finally, it is worth mentioning that every change performed in the application in the production environment will undergo a previous testing process and will include a clear justification stating the need for this change in production. All these changes will be performed following a formal change management methodology.

Conclusion

This paper shows that security in an application should be added from the first moment an idea to be developed arises. The guidelines proposed in this pages describe how this security addition should be performed. Finally, let us conclude stating that security in an application is an effective and efficient answer to real business requirements.

References

These books have inspired this paper and they have also been specifically referenced in this paper:

- [CRAMM]: CRAMM. "UK Government's Risk Analysis and Management Method (CRAMM). Countermeasures database". Version 3, 1996. Countermeasures divided into group and subgroup.
- [Cresson]: Cresson Wood, Charles. "Information security policies made easy version 6. A comprehensive set of information security policies". USA: Baseline Software, 1998. Several measures.
- [Krutz]: Krutz, Ronald L. Dean Vines, Russell. "The CISSP Prep Guide". New York: John Wiley & Sons, 2001. Pages 3-4,16-20.
- [TCSEC]: Trusted Computer System Evaluation Criteria, US Department of Defense, 1983.

Internet references

These Internet references have been read for the elaboration of this paper and specifically referenced:

- [Owasp]: Curphey, Mark et al. "A guide to building secure web applications". The open web application security project. 11 Sep 2002. URL: http://owasp.org. Document downloadable from URL: http://prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download (2 Jan 2003).
- [Herzog]: Herzog, Pete et al.Open Source Security Testing Methodology Manual. Institute for security and open methodologies (former ideahamster.org). URL: http://www.isecom.org/projects/osstmm.htm (3 Jan 2003).
- [Peteanu]: Peteanu, Razvan. "Best practices for secure development". Version 4.03, Oct 2001. URL: http://members.rogers.com/razvan.peteanu (5 Jan. 2003).
- [Brassine]: De la Brassine, Pierre: Web application security for managers. SANS Reading room. August 24, 2002. URL: http://www.sans.org/rr/appsec/managers.php (22 Jan 2003).
- [Maung]: Maung, Peter. Preparing for a web security review. SANS Reading room. May 29, 2001 URL: http://www.sans.org/rr/audit/web_review.php (24 Jan 2003)