Core Security Requirements

Confidentiality Requirements

Confidentiality requirements are those that address protection against the unauthorized disclosure of data or information that are either private or sensitive in nature. The classification of data (covered later in this chapter) into sensitivity levels is often used to determine confidentiality requirements. Data can be broadly classified into public and non-public data or information. Public data is also referred to as *directory* information.

Any non-public data warrants protection against unauthorized disclosure and software security requirements that provide such protection need to be defined in advance. Confidentiality protection mechanisms are depicted in *Figure 2.3.*

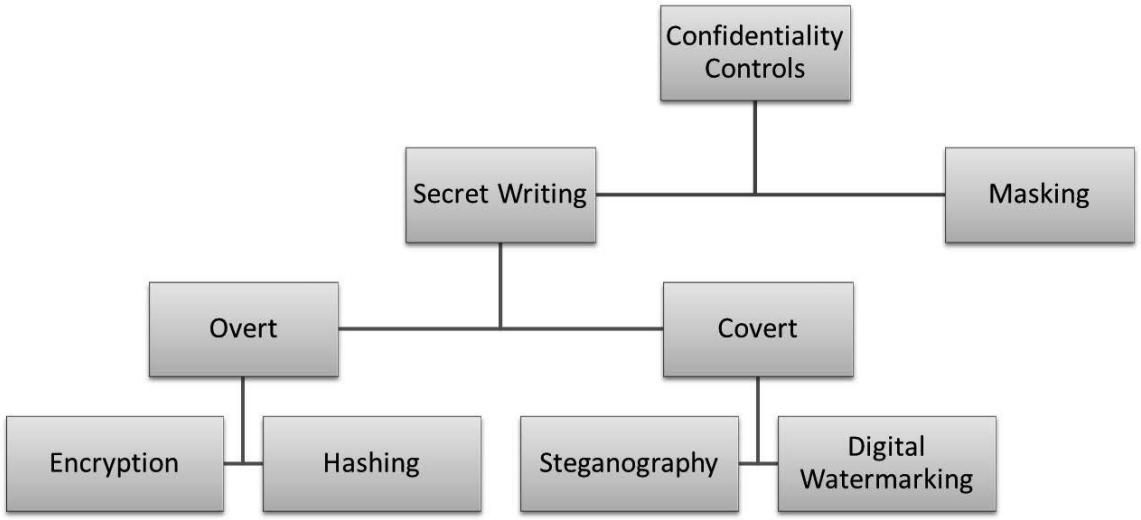

Figure 2.3 – **Confidentiality Protection Mechanisms**

Secret writing is a protection mechanism in which the goal is to prevent the disclosure of the information deemed secret. This includes overt cryptographic mechanisms such as encryption and hashing or covert mechanisms such as steganography and digital watermarking. The distinction between the overt and covert forms of secret writing lies in their objective to accomplish disclosure protection. The goal of overt secret writing is to make the information humanly indecipherable or unintelligible even if disclosed whereas the goal of covert secret writing is to hide information within itself or in some other media or form.

Overt secret writing, also commonly referred to as cryptography includes Encryption and Hashing. *Encryption* uses a bi-directional algorithm in which humanly readable information (referred to as clear text) is converted into humanly

unintelligible information (referred to as cipher text). The inverse of encryption is *decryption,* which is the process by which cipher text is converted into plain text. *Hashing* on the other hand is a one-way function where the original data or information that needs protection is computed into a fixed length output that is indecipherable. The computed value is referred to as a hash value, digest or hash sum. The main distinction between encryption and hashing is that unlike in encryption, the hashed value or hashed sum cannot be converted back to the original data and hence the one-way computation. So hashing is primarily used for integrity (non-alteration) protection, although it can be used as a confidentiality control, especially in situations when the information is stored and the viewers of that information should not be allowed to re-synthesize the original value by passing it through the same hashing function. A good example of this is when there is a need to store passwords in databases. Only the creator of the password should be aware of what it is. When the password is stored in the backend database, its hashed value should be the one that is stored. This way hashing provides disclosure protection against insider threat agents who may very well be the database administration within the company. When the password is used by the software for authentication verification, the user can supply their password, which is hashed using the same hashing function and then the hash values of the supplied password and the hash value of the one that is stored can be compared and authentication decisions can be accordingly undertaken.

The most common forms of covert secret writing are Steganography and Digital Watermarking. Steganography is more commonly referred to as *invisible ink* writing and is the art of camouflaging or hidden writing, where the information is hidden and the existence of the message itself is concealed. Steganography is primarily useful for covert communications and is useful and prevalent in military espionage communications. Digital watermarking is the process of embedding information into a digital signal. These signals can be audio, video, or pictures. Digital watermarking can be accomplished in two ways - visible and invisible. In visible watermarking, there is no special mechanism to conceal the information and it is visible to plain sight. This is of little consequence to us from a security standpoint. However, in invisible watermarking, the information is concealed within other media and the watermark is used to uniquely identify the originator of the signal, thereby making it possible for authentication purposes as well, besides confidentiality protection. Invisible watermarking is however mostly used for copyright protection, deterring and preventing unauthorized

copying of digital media. Digital watermarking can be accomplished using steganographic techniques as well.

Masking is a weaker form of confidentiality protection mechanism in which the original information is either asterisked or X'ed out. You may have noticed this in input fields that take passwords. This is primarily used to protect against shoulder surfing attacks, which are characterized by someone looking over another's shoulder and observing sensitive information. The masking of credit card numbers or social security numbers (SSN), except for the last four digits when printed on receipts or displayed on a screen is an example of masking providing confidentiality protection.

Confidentiality requirements need to be defined throughout the information life cycle from the origin of the data in question to its retirement. It is necessary to explicitly state confidentiality requirements for non-public data:

- In ***Transit***: When the data is transmitted over unprotected networks i.e., data-in-motion.
- In ***Processing***: When the data is held in computer memory or media for processing
- In ***Storage***: When the data is at rest, within transactional systems as well as non-transactional systems including archives i.e., data-at-rest.

Confidentiality requirements may also be *time bound*, i.e., some information may require protection only for a certain period of time. An example of this is news about a merger or acquisition. The date when the merger will occur is deemed sensitive and if stored or processed within internal Information Technology (IT) systems, it requires protection until this sensitive information is made public. Upon public press release of the merger having been completed, information deemed sensitive may no longer require protection as it becomes directory or public information. The general rule of thumb is that confidentiality requirements need to be identified based on the classification data is given and when that classification changes (say from sensitive to public), then appropriate control requirements need to be redefined.

Some good examples of confidentiality security requirements that should be part of the software requirements specifications are:

- "Personal health information must be protected against disclosure using approved encryption mechanisms."
- "Password and other sensitive input fields need to be masked."

- "Passwords must not be stored in the clear in backend systems and when stored must be hashed with at least an equivalent to the SHA-256 hash function."
- "Transport layer security (TLS) such as Secure Socket Layer must be in place to protect against insider man-in-the-middle (MITM) threats for all credit card information that is transmitted."
- "The use of non-secure transport protocols such as File Transfer Protocol (FTP) to transmit account credentials in the clear to third parties outside your organization should not be allowed."
- "Log files must not store any sensitive information as defined by the business in humanly readable or easily decipherable form."

As we determine requirements for ensuring confidentiality in the software we build or acquire, we must take into account the timeliness and extent of the protection required.

Integrity Requirements

Integrity requirements for software are those security requirements that address two primary areas of software security *viz.* reliability assurance and protection or prevention against unauthorized modifications. Integrity refers not only to the system or software modification protection (system integrity) but also the data that the system or software handles (data integrity). When integrity protection assures *reliability*, it essentially refers to ensuring that the system or software is functioning as it is designed and expected to. In addition to reliability assurance, integrity requirements are also meant to provide security controls that will ensure that the *accuracy* of the system and data is maintained. This means that data integrity requires that information and programs be changed only in a specified and authorized manner by authorized personnel. While integrity assurance primarily addresses the reliability and accuracy aspects of the system or data, it must be recognized that integrity protection also takes into consideration the *completeness* and *consistency* of the system or data that the system handles.

Within the context of software security, we have to deal with both system and data integrity. Injection attacks such as SQL injection that makes the software act or respond in a manner not originally designed to is a classic example of system integrity violation. Integrity controls for data in transit or data at rest need to provide assurance against deliberate or inadvertent unauthorized manipulations. The requirement to provide assurance of integrity needs to be defined explicitly in the software requirements specifications.

Security controls that provide such assurance include input validation, parity bit checking and cyclic redundancy checking (CRC) and hashing. *Input validation* provides a high degree of protection against injection flaws and provides both system and data integrity. Allowing only valid forms of input to be accepted by the software for processing mitigates several security threats against software (covered in the secure software implementation chapter). In the secure software design and secure software implementation chapters, we will cover input validation in depth and the protections it provides. *Parity bit checking* is useful in the detection of errors or changes made to data when it is transmitted. Mathematically, parity refers to the evenness or oddness of an integer. A parity bit (0 or 1) is an extra bit that is appended to a group of bits (byte, word or character) so that the group of bits will either have an even or odd number of 1's. The parity bit is 0 (even) if the number of 1's in the input bit stream is even and 1 (odd) if the number of 1's in the input bit stream is odd. Data integrity checking is performed at the receiving end of the transmission, by computing and comparing the original bit stream parity with the parity information of the received data. A common usage of parity bit checking is to do a *cyclic redundancy check (CRC)* for data integrity as well, especially for messages longer than one byte (8 bits) long. Upon data transmission, each block of data is given a computed CRC value, commonly referred to as a *checksum.* If there is an alteration between the origin of data and its destination, the checksum sent at the origin will not match with the one that is computed at the destination. Corrupted media (CD's, DVDs) and incomplete downloads of software yield CRC errors. The checksum is the end product of a non-secure hash function. *Hashing* provides the strongest forms of data integrity. Although, hashing is mainly used for integrity assurance, it can also provide confidentiality assurance as we covered earlier in this chapter.

Some good examples of integrity security requirements that should be part of the software requirements specifications are:

- "All input forms and Querystring inputs need to be validated against a set of allowable inputs before the software accepts it for processing."
- "Software that is published should provide the recipient with a computed checksum and the hash function used to compute the checksum, so that the recipient can validate its accuracy and completeness."

- "All non-human actors such as system and batch processes need to be identified, monitored and prevented from altering data as it passes on systems that they run on, unless explicitly authorized to."

As we determine requirements for ensuring integrity in the software we build or acquire, we must take into account the reliability, accuracy, completeness and consistency aspects of systems and data.

Availability Requirements

Although the concept of availability may seem to be more closely related to business continuity or disaster recovery disciplines than it is to security, it must be recognized that improper software design and development can lead to destruction of the system/data or even cause Denial of Service (DoS). It is, therefore, imperative that availability requirements are explicitly determined to ensure that there is no disruption to business operations. Availability requirements are those software requirements that ensure the protection against destruction of the software system and/or data, thereby assisting in the prevention against DoS to authorized users. When determining availability requirements, the Maximum Tolerable Downtime (MTD) and Recovery Time Objective (RTO) must both be determined. MTD is the measure of the maximum amount of time that the software can be in a state of not providing expected service. In other words, it is the measure of the minimum level of availability that is required of the software for business operations to continue without unplanned disruptions as per expectations. But since all software fails or will fail eventually, in addition to determining the MTD, the RTO must also be determined. RTO is the amount of time by which the system or software needs to be restored back to the expected state of business operations for authorized business users, when it goes down. Both MTD and RTO should be explicitly stated in the Service Level Agreements (SLA). MTD are sometimes are referred to as Maximum Tolerable Period of Disruption (MTPD) as the system may cause disruptions and not downtime to the business. Recovery Point Objective (RPO) also expressed in units of time is the maximum allowed data or productivity loss when the system becomes disrupted or down. It is the point in time to which the disaster recovery personal plans to recover the system to. There are several ways to determine availability requirements for software. These methods include determining the adverse effects of software downtime through Business Impact Analysis (BIA) or stress and performance testing.

BIA must be conducted to determine the adverse impact that the unavailability of software will have on business operations. This may be

measured quantitatively such as loss of revenue for each minute the software is down, cost to fix and restore the software back to normal operations or fines that are levied on the business upon software security breach. It may also be qualitatively determined which include loss of credibility, confidence or loss of brand reputation. In either case, it is imperative to include the business owners and end-users to accurately determine MTD and RTO as a result of the BIA exercise. BIA can be conducted for both new and existing versions of software. In situations when there is an existing version of the software, then stress and performance test results from the previous version of the software can be used to ensure high availability requirements are included for the upcoming versions as well. *Table 2.1* tabulates the downtime that will be allowed for a percentage of availability that is usually measured in units of nines. Such availability requirements and planned downtime amounts must be determined and explicitly stated in the SLA and incorporated into the software requirements documents.

Measurement	Availability %	Downtime per Year	Downtime per Month	Downtime per Week
Three Nines	99.9%	8.76 hours	43.2 minutes	10.1 minutes
Four Nines	99.99%	52.6 minutes	4.32 minutes	1.01 minutes
Five Nines	99.999%	5.26 minutes	25.9 seconds	6.05 seconds
Six Nines	99.9999%	31.5 seconds	2.59 seconds	0.605 seconds

Table 2.1 – **High availability requirements as measures of Nines**

In determining availability requirements, understanding the impact of failure due to a breach of security is vitally important. Insecure coding constructions such as dangling pointers, improper memory de-allocations and infinite loop constructs can all impact availability and when requirements are solicited, these repercussions of insecure development must be identified and factored in. End-to-end configuration requirements ensure that there is no single point of failure and this should be part of the software requirements documentation. In addition to end-to-end configuration requirements, load balancing requirements need to be identified and captured as well.

Some good examples of availability requirements that have a bearing on software security are given below and should be part of the software requirements specifications.

- "The software shall ensure high availability of five nines (99.999%) as defined in the SLA."

- "The number of users at any one given point of time who should be able to use the software can be up to 300 users."
- "Software and data should be replicated across data centers to provide load balancing and redundancy."
- "Mission critical functionality in the software should be restored to normal operations within 1 hour of disruption; mission essential functionality in the software should be restored to normal operations within 4 hours of disruption; and mission support functionality in software should be restored to normal operations within 24 hours of disruption."

Authentication Requirements

The process of validating an entity's claim is authentication. The entity may be a person, a process or a hardware device. The common means by which authentication occurs is that the entity provides identity claims and/or credentials which are validated and verified against a trusted source holding those credentials. Authentication requirements are those that verify and assure the legitimacy and validity of the identity that is presenting entity claims for verification.

In the secure software concepts domain, we learned that authentication credentials could be provided by different factors or a combination of factors that include knowledge, ownership or characteristics. When two factors are used to validate an entity's claim and/or credentials, it is referred to as *two-factor* authentication and when more than two factors are used for authentication purposes, it is referred to as *multi-factor* authentication. It is important to determine first, if there exists a need for two- or multi-factor authentication. It is also advisable to leverage existing and proven authentication mechanisms and requirements that call for custom authentication processes should be closely reviewed and scrutinized from a security standpoint so that no new risks are introduced in implementing custom and newly developed authentication validation routines.

There are several means by which authentication can be implemented in software. Each has its own pros and cons as it pertains to security. In this section, we cover some of the most common forms of authentication. However, depending on the business context and needs, authentication requirements need to be explicitly stated in the software requirements document so that when the software is being designed and built, security implications of those authentication requirements can be determined and addressed accordingly. The most common forms of authentication are

- Anonymous
- Basic
- Digest
- Integrated
- Client certificates
- Forms
- Token
- Smart cards
- Biometrics

Anonymous Authentication:

Anonymous authentication is the means of access to public areas of your system without prompting for credentials such as username and password. As the name suggests, anyone even anonymous users are allowed access and there is no real authentication check for validating the entity. Although this may be required from a privacy standpoint, the security repercussions are serious since with anonymous authentication there is no way to link a user or system to the actions they undertake. This is referred to as *unlinkability* and if there is no business need for anonymous authentication to be implemented, it is best advised to avoid it.

Basic Authentication:

One of the HyperText Transport Protocol (HTTP) 1.0 specifications is basic authentication which is characterized by the client browser prompting the user to supply their credentials. These credentials are transmitted in Base-64 encoded form. Although this provides a little more security than anonymous authentication, Basic authentication must be avoided as well, since the encoded credentials can be easily decoded.

Digest Authentication:

Digest authentication is a challenge/response mechanism, which unlike Basic authentication, does not send the credentials over the network in clear text or encoded form, but instead sends a message digest (hash value) of the original credential. Authentication is performed by comparing the hash values of what was previously established and what is currently supplied as an entity claim. Using a unique hardware property, that cannot be easily spoofed, as an input (salt) to calculate the digest, provides heightened security, when implementing Digest authentication.

Integrated Authentication:

Commonly known as NTLM authentication or NT challenge/response authentication, like Digest authentication, the credentials are sent as a digest. This can be implemented as a standalone authentication mechanism or in conjunction with Kerberos v5 authentication when delegation and impersonation is necessary in a trusted sub-system infrastructure. Wherever possible, especially in intranet settings, it is best to use integrated authentication since the credentials are not transmitted in clear text and it is efficient in handling authentication needs.

Client Certificate-Based Authentication:

Client certificate-based authentication works by validating the identity of the certificate holder. These certificates are issued to organizations or users by a certification authority (CA) that vouches for the validity of the holder. These certificates are usually in the form of digital certificates and the current standard for digital certificates is ITU X.509 v3. If you trust the CA and you validate that the certificate that is presented for authentication has been signed by the trusted CA, then you can accept the certificate and process access requests. These are particularly useful in an Internet/ecommerce setting, when you cannot implement integrated authentication across your user base. Digital certificates is covered in more detail in the Secure Software Design chapter.

Forms Authentication:

Predominantly observed in web applications, Forms authentication requires the user to supply a username and password for authentication purposes and these credentials are validated against a directory store which can be the active directory, a database or configuration file. Since the credentials collected are supplied in clear text form, it is advisable to first cryptographically protect the data being transmitted in addition to implementation transport layer security (TLS) such as SSL or network layer security such as IPSec. *Figure 2.4* illustrates an example of a username and password login box used in Forms authentication.

Figure 2.4 – **Forms Authentication**

Token-Based Authentication:

The concept behind token based authentication is pretty straightforward. It is usually used in conjunction with Forms authentication where a username and password is supplied for verification. Upon verification, a token is issued to the user who supplied the credentials. The token is then used to grant access to resources that are requested. This way the username and password need not be passed on each call. This is particularly useful in single sign on (SSO) situations. While Kerberos tickets are restricted to the domain they are issued, Security Assertion Markup Language (SAML) tokens which are XML representations of claims an entity makes about another entity is considered the *de facto* token in cross-domain federated SSO architectures. We will cover SSO in more detail in the Secure Software Design chapter.

Smart Cards-Based Authentication:

Smart cards provide ownership (something you have) based authentication. They contain a programmable embedded microchip that is used to store authentication credentials of the owner. The security advantage that smart cards provide is that they can thwart the threat of hackers stealing authentication credentials from a computer, since the authentication processing occurs on the smart card itself. However, a major disadvantage of smart cards is that the amount of information that can be stored is limited to the size of the microchip's storage area and cryptographic protection of stored credentials on the smart card is limited as well.

One Time (dynamic) passwords (OTP) provide the maximum strength of authentication security and OTP tokens (also known as key fobs) require two factors, knowledge (something you know) and ownership (something you have). These tokens dynamically provide a new password at periodic intervals. Like token based authentication, the user enters the credential information they know and is issued a PIN that is displayed on the token device such as a Radio Frequency Identification (RFID) device they own. Since the PIN is not static and dynamically changed every few seconds, it makes it virtually impossible for a malicious attacker to steal authentication credentials.

Biometric Authentication:

This form of authentication uses biological characteristics (something you are) for providing the identity's credentials. Biological features such as retinal blood vessel patterns, facial features and fingerprints are used for identity verification purposes. Since biological traits can potentially change over time due to aging

or pathological conditions, one of the major drawbacks of biometric based authentication implementation is that the original enrollment may no longer be valid and this can yield to DoS to legitimate users. This means that authentication workarounds need to be identified, defined and implemented in conjunction to biometrics, and these need to be captured in the software requirements. The FIPS 201 Personal Identity Verification standard provides guidance that the enrollment data in systems implementing biometric based authentication needs to be changed periodically.

Additionally biometric authentication requires physical access which limits its usage in remote access settings.

Errors observed in biometric based authentication systems are of two types viz. Type I error and Type II error. Type I error is otherwise known as False Rejection error where a valid and legitimate enrollee is denied (rejected) access. It is usually computed as a rate and is referred to as False Rejection Rate (FRR). Type II error is otherwise known as False Acceptance error where an imposter is granted (accepted) access. This is also computed as a rate and is referred to as False Acceptance Rate (FAR). The point at which the FRR equals the FAR is referred to as the Crossover Error Rate (CER) as depicted in *Figure 2.5*. CER is primarily used in evaluating different biometric devices and technologies. Devices which assure more accurate identity verification are characterized by having a low Crossover Error Rate.

Some good examples of authentication requirements that should be part of the software requirements specifications are:

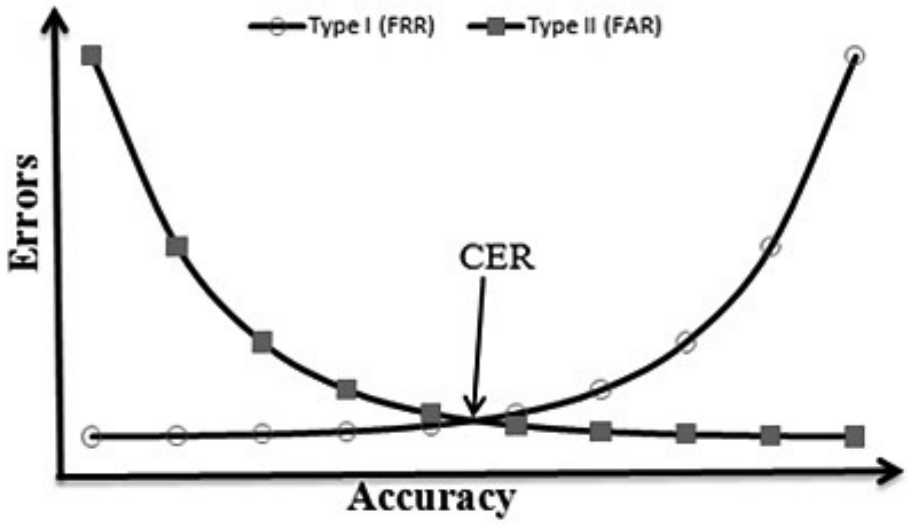

Figure 2.5 – **Crossover Error Rate**

- "The software will be deployed only in the intranet environment and the authenticated user should not have the need to provide username and password once they have logged on to the network."
- "The software will need to support single sign on with 3rd party vendors and suppliers that are defined in the stakeholder list."
- "Both intranet and Internet users should be able to access the software."
- "The authentication policy warrants the need for two- or multi-factor authentication for all financially processing software."

Identifying the proper authentication requirements during the early part of the SDLC helps to mitigate many serious security risks at a later stage. These need to be captured in the software specifications so that they are not overlooked when designing and developing the software.

Authorization Requirements

Layered upon authentication, authorization requirements are those that confirm that an authenticated entity has the needed rights and privileges to access and perform actions on a requested resource. These requirements answer the question as to what one is allowed or not allowed to do. To determine authorization requirements, it is important to first identify the *subjects* and *objects*. Subjects are the entities that are requesting access and Objects are the items that subject will act upon. A subject can be a human user or a system process. Actions on the objects also need to be explicitly captured. Actions as they pertain to data or information that the user of the software can undertake are commonly referred to as CRUD operations, which stand for Create, Read, Update or Delete data. Later in this chapter, we shall cover subject-object modeling in more detail as one of the mechanisms to capture authorization requirements.

Access control models are primarily of the following types

- Discretionary Access Control (DAC)
- Non-Discretionary Access Control (NDAC)
- Mandatory Access Control (MAC)
- Role-Based Access Control (RBAC)
- Resource-Based Access Control

Discretionary Access Control (DAC)

DAC is defined as "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in