Certified Tester
Security Test Engineer Syllabus

Version 1.0.1

International Software Testing Qualifications Board

Copyright Notice

Revision History

Version	Date	Remarks
1.0	18-10-2024	Final version
1.0.1	01-31-2025	Updated and final version after technical review

Table of Contents

Acknowledgements

This document was formally released by the General Assembly of the ISTQB® on 10-18-2024

It was produced by the security test task force from the International Software Test Qualifications Board: Dr. Frank Simon (chair), Alain Ribault, Gabriel Firmino Barjollo, Michael Pott, Beata Karpinska, Maria Kispal, Frans Dijkman

The following persons participated in the reviewing, commenting and balloting of this syllabus:

Attila Toth, Claude Zhang, Dani Almog, Daniel Säther. Gary Mogyorodi, Haiying Liu (Sandy), Ingvar Nordström, Laura Albert, Nancy Thompson, Petr Zacek, Szilard Szell, Tal Pe'er, Tamás Gergely, Claude Zhang, Daniel van der Zwan, Dingguofu, Giancarlo Tomasig, Klaus Skafte, Markus Niehammer, Mattijs Kemmink, Meile Posthuma, Michael Stahl, Nicola Rosa, Samuel Ouko, Tauhida Parveen, Tetsu Nagata

0. Introduction

0.1. Purpose of this Syllabus

This syllabus forms the basis for the International Software Test Qualification Security Test Engineer. The ISTQB® provides this syllabus as follows:

1. To member boards, to translate into their local language and to accredit training providers. Member boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To certification bodies, to derive examination questions in their local language adapted to the learning objectives for this syllabus.
3. To training providers, to produce courseware and determine appropriate teaching methods.
4. To certification candidates, to prepare for the certification exam (either as part of a training course or independently).
5. To the international software and systems engineering community, to advance the profession of software and systems test, and as a basis for books and articles.

0.2. Business Outcomes

This section lists the Business Outcomes expected of a candidate who has achieved the Security Test Engineer Specialist Level certification.

A Security Test Engineer Certified Tester can …

STE-BO1	Understand the fundamental security paradigms, and their impact on security testing
STE-BO2	Use and apply appropriate security test techniques and know their strengths and limitations
STE-BO3	Contribute to planning, designing, and executing security tests
STE-BO4	Understand how security testing standards and security best practices can be utilized for security testing
STE-BO5	Adjust and perform security testing activities accordingly to specific organization context
STE-BO6	Adjust and perform security testing activities accordingly to specific development methods and software development lifecycles
STE-BO7	Feed security testing results into an information security management system (ISMS) for an active security risk management

STE-BO8	Collect, evaluate, and aggregate test results, and write a detailed test report which includes all evidence and findings
STE-BO9	Based on a required security testing approach, identify proper requirements for tooling, and assist in the selection of security testing tools

0.3. Examinable Learning Objectives and Cognitive Level of Knowledge

Learning objectives support the business outcomes and are used to create the Certified Tester Security Test Engineer Level exams.

In general, all contents of this syllabus are examinable at a K1 level. That is, the candidate may be asked to recognize, remember, or recall a keyword or concept mentioned in any of the nine chapters. The specific learning objectives levels are shown at the beginning of each chapter, and classified as follows:

- K1: Remember
- K2: Understand
- K3: Apply
- K4: Analyze

Further details and examples of learning objectives are given in Appendix A.

All terms listed as keywords and security domain-specific keywords shall be examinable (K1), even if they are not explicitly mentioned in the learning objectives.

0.4. The Security Test Engineer Certification Exam

The Security Test Engineer certification exam will be based on this syllabus. The other syllabus Security Test Analyst focuses on designing security tests that are later executed by the Security Test Engineer.

Answers to exam questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable, except for the Appendices. Standards and books are included as references, but their content is not examinable, beyond what is summarized in the syllabus itself from such standards and books.

Refer to Exam Structures and Rules document for the Security Test Engineer document for further details.

The entry criterion for taking the Security Test Engineer exam is that candidates have an interest in software testing. However, it is recommended that candidates also have at least a minimal background in either software development, software testing or security testing.

Entry Requirement Note: The ISTQB® Foundation Level certificate shall be obtained before taking the Security Test Engineer certification exam.

Completion of an accredited training course is not a pre-requisite for the exam.

0.5. Accreditation

An ISTQB® Member Board may accredit training providers whose course material follows this syllabus. Training providers should obtain accreditation guidelines from the Member Board or body that performs the accreditation. An accredited course is recognized as conforming to this syllabus and is allowed to have an ISTQB® exam as part of the course.

The accreditation guidelines for this syllabus follow the general Accreditation Guidelines published by the Processes Management and Compliance Working Group.

0.6. Handling of Standards

Standards are referenced in the Security Test Engineer Syllabus (e.g., NIST and ISO). The purpose of these is to provide a framework or to provide a source of additional information if desired by the reader. Please note that the syllabus uses standards as a reference, and they are not intended for examination. Refer to chapter 4 for more information on the use of standards, best practices and norms.

0.7. Level of Detail

The level of detail in this syllabus allows for internationally consistent courses and exams. In order to achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the Specialist Level
- A list of terms (keywords) that students must be able to recall
- A list of domain-specific keywords associated with security that students must be able to recall
- Learning objectives for each knowledge area, describing the cognitive learning outcomes to be achieved
- A description of the key concepts, including references to recognized sources

The syllabus content is not a description of the entire knowledge area of security testing; it reflects the level of detail to be covered in Security Test Engineer training courses at Specialist level. It focuses on test concepts and techniques that can be applied to all software projects independent of the software development lifecycle employed.

0.8. How this Syllabus is Organized

There are nine chapters with examinable content. The top-level heading for each chapter specifies the training time for the chapter; timing is not provided below chapter level. For accredited training courses, the syllabus requires a minimum of 1290 minutes (about 22 hours) of instruction, distributed across the nine chapters as follows:

- Chapter 1: Security Paradigms – 135 minutes
- Chapter 2: Security Test Techniques - 150 minutes

- Chapter 3: The Security Test Process - 120 minutes
- Chapter 4: Security Testing Standards and Best Practices - 195 minutes
- Chapter 5: Adjusting Security Testing to the Organizational Context - 195 minutes
- Chapter 6: Adjusting Security Testing to Software Development Lifecycle Models - 165 minutes
- Chapter 7: Security Testing as Part of an Information Security Management System (ISMS) - 105 minutes
- Chapter 8: Reporting Security Test Results - 135 minutes
- Chapter 9: Security Testing Tools – 90 minutes

1. Security Paradigms – 135 minutes (K3)

Keywords

availability, confidentiality, integrity, security testing, vulnerability

Security Keywords

open-source software, zero-trust

Learning Objectives for Chapter 1:

1.1 Asset Security Levels

STE-1.1.1 (K2) Explain different security levels of assets and their corresponding protection level

STE-1.1.2 (K2) Explain the relationship between information sensitivity and security testing

1.2 Security Audits

STE-1.2.1 (K2) Describe the role of security testing in the context of security audits

1.3 The Concept of Zero Trust

STE-1.3.1 (K2) Explain the concept of zero trust

STE-1.3.2 (K3) Apply the zero trust in security testing

1.4 Open-Source Software

STE-1.4.1 (K2) Exemplify the concept of open-source software reuse in software development and its impact on security testing

1.1. Asset Security Levels

An asset is anything that has value in an organization and which, therefore, requires protection. Assets enable organizations to operate business processes and make decisions. Every information and data asset is vulnerable and thus should be protected. Assets are objects of information security. Assets can be people, information, software, hardware, functions, processes, and corporate reputation facilities [Chapple 2021]. Some examples:

- software assets: operating system, applications, and databases
- information assets: business plans, documentation, inventions, pictures, and personal records
- hardware assets: computer systems, data storage, and data communication devices
- physical assets: facilities

1.1.1. Assets and Their Corresponding Protection Level

The value of an asset is determined by three pillars of information security (called the CIA triad): confidentiality, integrity, and availability [Stallings18].

Confidentiality seeks to prevent the unauthorized disclosure of information. A loss of confidentiality occurs when information is disclosed to an unauthorized party or system [Stallings18].

Integrity seeks to prevent data from being modified or deleted by an unauthorized party. [Stallings18].

Availability ensures that information is available when needed. [Stallings18].

The three pillars of information security can be classified into the following levels: high, medium, and low. The higher the security level, the higher the requirement for the protection level (safeguards) to be deployed. Safeguards are such as security functions and constraints, personnel security, and the security of physical structures, areas, and devices.

The loss of confidentiality, integrity, or availability can have an impact on organizational operation, organizational assets, people, customers or even countries.

Asset classification defines asset sensitivity and confidentiality levels. This helps organizations to implement appropriate security controls and protection levels.

If the asset has low sensitivity, a protection level might be set to low security level, and security testing might not be necessary. The pillar suited for this is availability.
An example of low sensitivity can be when the availability of information is of greater importance to a mass of people such as traffic information.

If the asset has medium sensitivity, a protection level will be set to medium security level, and security testing will be necessary. The pillar suited for this is integrity.
An example of medium sensitivity can be when people need accurate information from trusted sources such as authorities.

If the asset has high sensitivity, a protection level will be set to a high security level, and security testing will be a must. The pillar suited for this is confidentiality. An example of high sensitivity can be personal information about employees.

1.1.2. Information Sensitivity and Security Testing

Information sensitivity represents the degree to which information requires protection to ensure its confidentiality, integrity, and availability [NIST Glossary]. Since the consequences of breaching sensitive information can range from minor to disastrous, a security test must be done before any breach occurs.

The purpose of security testing is to verify that an implementation protects data and maintains functionality as intended. The more protection an asset needs, the more security actions are needed. Security testing helps to identify weaknesses and vulnerabilities, and it checks if proper security controls are implemented.

Security testing cannot guarantee that a system or organization will be free from vulnerabilities. Such steps include performing security testing to achieve the following objectives:

- Evaluating the effectiveness of existing security controls
- Discovering weaknesses and vulnerabilities
- Establishing a security test strategy which includes confirmation tests for tracking the progress of any software patches and long-term upgrades

For the Security Test Engineer (STE), a security risk assessment done from an information sensitivity perspective can be a rich source of information from which security tests can be planned and designed. In addition, a security risk assessment can be used to prioritize security tests such that risks and priorities can be determined and those with the highest risk levels are targeted for more rigorous testing. Risk assessment is only a snapshot at the current point in time and may be based on limited information.

1.2. Security Audits

A security audit is an independent review and examination of the security of an organization's information system by controlling how well it conforms to an established set of criteria. The audits are intended to determine the adequacy of system controls, ensure compliance with an established security policy and procedures. But also, to detect breaches in security services, and recommend any changes that are indicated for countermeasures [NIST Glossary].

1.2.1. Security Audits and Security Testing

Security audits have the following characteristics:

- They can be performed by internal or external auditors.
- They focus on aspects of an organization's security processes and controls, such as:
    - Physical components of the information system and the environment in which the information is stored
    - Applications and software, including security patches and configurations
    - Controls for user rights and privileges
    - Network vulnerabilities, including evaluations of the connection between different points within and outside the organization's network
    - How employees collect, share, and store information
    - Intrusion detection mechanisms
    - Response plans in the event of a breach

- They are a type of static testing (see section 2.1.2) which involves manual examination of work products or automated reviews with security audit tools.
- They investigate aspects of an organization's security policies, procedures, and controls which are difficult to test dynamically.
- They check the effectiveness of installed security controls and identify where the criteria set by the organization have or have not been achieved at a particular point in time.
- They do not guarantee all vulnerabilities will be found but provide assurance that problem areas are identified and indicate where remedial action is needed.

Security audits should be part of the regular routine. They can be done as:

- Onetime assessments for special circumstances, such as when an organization introduces a new software platform or a new integration. A security test and audit should be performed to discover any new risks and/or defects.
- Regularly scheduled audits to verify that security processes and procedures are being followed and that they are adequate for the current business climate and needs.

In some security audit approaches, security testing is performed as part of the audit process to determine whether security controls are actually in place and working effectively. However, the scope of a security audit is much larger than security testing.

Security testing and auditing work together. Auditing identifies deficiencies and areas of importance to test. Security testing is the means by which it is proven or disproven that the security controls are actually in place and working effectively.

1.3. The Concept of Zero Trust

1.3.1. What is Zero Trust?

Zero trust is a security model created out of a collection of concepts and ideas designed to minimize uncertainty in enforcing accurate, least privilege per-request access. It is based on the principle of strict access controls and not trusting anyone by default, even if everyone is already inside the network perimeter.

The zero trust model embodies a "trust nothing, verify everything, the face of a network is viewed as compromised" principle.

Increased use of online services has resulted in a corresponding increase in vulnerabilities and attacks. The information is often spread across cloud vendors, which has resulted in perimeter-based security concepts becoming less effective in providing security for organizations, employers, users, and customers. Traditional perimeter-based security trusts anyone and anything inside the network. The problem with this approach is that once an attacker gains access to the network, they have access to all the assets inside.

Using the zero trust model, information systems and services operate under the assumption that their networks are already compromised and that the network has no trusted space. The zero trust perspective causes the practice that moves security defenses from static, network-based perimeters to focus on users, assets, and resources [NIST Glossary].

Benefits of zero trust [Cloudflare]:

- Reduces an organization's attack surface
- Minimizes the damage of an attack by restricting the breach to a small area and lowers the cost of recovery
- Reduces the impact of user credential theft and phishing attacks by requiring multiple authentication factors

1.3.2. Zero Trust in Security Testing

Today's networks are perimeterless, migrating from flat deployments into dynamic, distributed, and hybrid environments. Organizations adapt to the growing complexity of their environments, which embraces hybrid workplaces, remote workers, interactions with other organizations and suppliers, and needs to protect people, devices, applications, networks, and data wherever they might be located. The zero trust model with the assumption "trust nothing, verify everything", drives the need for a complete paradigm shift in IT security where people, devices, applications, networks and data must undergo stringent validations before gaining access to an application or requested resource.

Zero trust principles [Micro22] [Cloudflare] include:

- Continuous monitoring and verification: always verify access for all resources. Logins and connections time out periodically, forcing users and devices to be continuously reverified.
- "Least privilege access" principle: give users only as much access as they need.
- Multifactor authentication: requires more than one piece of evidence to authenticate a user. Just entering a password is not enough to allow access.
- Security monitoring of device endpoints accessing data, such as laptops, workstations, tablets, mobile devices: every remote endpoint can be the entry point for an attack.
- Microsegmentation: divide security perimeters into small zones to achieve separate access for separate parts of the network.
- No lateral movement within networks without continuous validation: zero trust access is segmented and must be re-established periodically. An attacker cannot move across to other microsegments within the network. Once the attacker's presence is detected, the compromised segment or user account can be quarantined, and cut off from further access.
- Protection of the data across the files and content: encryption and access restrictions based on organizational policies.

The zero trust model affects how security testing should be managed. This means focusing test cases on zero trust security mechanisms. Security testing contributes to identifying the following possible weaknesses:

- Network: Zero trust microsegments, reducing damage and highlighting violations
  - Test for cross segment traffic with automation and with user defined microsegments.
- Data: Zero trust requires that data shall be transmitted in a secure encrypted way.
  - Test for endpoints that expose or store data using unencrypted methods.
- Identity: people, devices and processes can only do what they are allowed to do.
  - Test if people, devices and/or processes have more than the necessary access permissions that may compromise the assets in the network.
  - Check if unauthorized users can access segments of the network resources.
- Devices should run only secure software and be centrally monitored and managed.
  - Test if the device is protected with security software and perform penetration testing.
- Limitation of the blast radius:
  - Tests are performed on a regular basis to show weaknesses to mitigate risk impact to limit the blast radius in case an external or internal breach occurs.

1.4. Open-Source Software (OSS)

OSS is developed and maintained via open collaboration, and is available, typically at no cost, for anyone to use, change and redistribute however they like.

1.4.1. The Concept of OSS and its Impact on Security Testing

OSS is built on open-source code to be freely used, modified, and shared by anyone. OSS is often distributed under licenses that comply with the definition of open-source, as provided by the Open-Source Initiative.

The main benefit of using OSS is code transparency and the assumption that many volunteer developers have checked the code to detect and resolve defects. The assumption is that this openness will lead to more people being involved in quickly identifying vulnerabilities and fixing defects.

However, the fact that these applications, libraries and other reusable objects are available all over the world presents a challenge, since anyone can update the code and potentially introduce vulnerabilities and attack vectors [Shacklett]. OSS generally has more attack vectors than closed (proprietary) software, because anyone can add intentional backdoors, propagate vulnerabilities through reuse, and exploit publicly disclosed vulnerabilities and human errors. Once a vulnerability has been published, the users of that version of the software are at risk of an attack.

Using OSS means that every published exploit for a specific component could potentially affect thousands of systems. When source code is available in executable versions, observation, reverse engineering, code reviews, disassembly and exploratory testing may be able to find vulnerabilities [Stallings18].

The STE performs the following tasks:

- Identifying open-source vulnerabilities

- Performing code reviews as part of shift left which targets the detection of defects early in the development process

When it comes to testing OSS for application security, the STE thinks like an attacker. Test cases capture how an application behaves under different use and misuse scenarios and enables developers to put proper risk mitigations in place.

Code reviews are performed by developers and STEs on both the producer's and consumer's sides to identify unsecured code.

The Open Web Application Security Project (OWASP) has made automated vulnerability detection tools available that are free to open-source projects [OWASP Top 10]. NIST offers guidance for OSS security [NIST SP 800-161].

2. Security Test Techniques - 150 minutes (K3)

Keywords

destructive testing, fuzz testing, malware scanning, vulnerability scanning

Security Keywords

authentication, authorization, encryption, firewall

Learning Objectives for Chapter 2:

2.1 Applying Security Test Types According to a Test Context

STE-2.1.1 (K2) Give examples for security test types according to a black-box, white-box and grey-box security context

STE-2.1.2 (K2) Give examples for security test types according to static security testing or dynamic security testing

2.2 Applying Security Test Types According to a Project and Technical Context

STE-2.2.1 (K3) Apply security test cases, based on a given security test approach, along with identified functional and structural security risks

STE-2.2.2 (K2) Describe how to test reconciliation and recertification for identities and permissions

STE-2.2.3 (K2) Describe how to test identity and access management control

STE-2.2.4 (K2) Describe how to test data protection control

STE-2.2.5 (K2) Describe how to test protective technologies

2.1. Applying Security Test Types According to a Test Context

2.1.1. Black-Box Testing, White-Box Testing, and Grey-Box Testing

Test types for the test basis are classified as black-box testing, grey-box testing and white-box testing [ISTQB FL]. The ISTQB Glossary [ISTQB Glossary] defines black-box testing and white-box testing.

Grey-box security testing is defined in [NIST] as “a test methodology that assumes some knowledge of the internal structure and implementation detail of the assessment object”. The STE has access to some information about the system under test (SUT) (e.g., a subset of a network addressing map, a subset of architecture documentation, a user access, and an access to an internal machine).

The STE has access to all needed information about the SUT during security testing:

- The architecture of the SUT
- The source code
- Data flows in the SUT
- Network design and zone structure
- Password requirements
- Firewall rules
- Authentication
- Log storage and management information

The choice of test techniques is based on the objectives of the security test approach as well as the availability of code. The STE should decide on the level of depth of the tests and whether to focus on external or internal threats.

2.1.2. Static and Dynamic Security Testing

Both static testing and dynamic testing are used in security testing to secure the SUT throughout its entire lifecycle.

Static Security Testing

From the STE's perspective, among the work products that can be reviewed during static security testing are:

- Security risk analysis documentation
- Security requirements

When looking for requirements gaps, the following security mechanisms should be considered:

- User Management
- Authentication
- Authorization
- Confidentiality
- Integrity
- Accountability
- Session management
- Transport security
- Tiered system segregation
- Legislative and standard compliance including privacy, government and industry standards

- Technical architectural security documentation
- Source code
- Configuration setup and infrastructure/operational setup

Dynamic Security Testing

Compared to static security testing, the aim of dynamic security testing is to check that the SUT correctly implements and uses security functions as required or specified and that these security functions cannot be bypassed while using the system or application.

Dynamic security testing can be performed by:

- Using black-box testing to evaluate security functions by checking that the test results are as expected when particular inputs are submitted
- Penetration testing: Tries to find vulnerabilities that could be exploited

It is recommended to execute dynamic security testing in the following manner in the following environments, if possible:

- The development environment
  This is the first opportunity for the STE to execute security tests. In this environment, the STE verifies that the security implementation conforms to security requirements (e.g., the correct development of a password by controlling the minimum size, the maximum size, and the mandatory types of characters).
- The test environment and the pre-production environment
  Often known as a staging or acceptance test environment, this environment should be as similar to the production environment as possible and should contain all the intended security measures that will be applied in the production environment.
- The production environment
  This is the most critical environment. The STE must take care not to disable the SUT and should conduct security audits to keep the system secure. The objective of security testing on the production environment is to check that newly discovered and known vulnerabilities are fixed.

Using a dynamic application security test (DAST) tool enables conditions to be detected that might lead to vulnerabilities (see chapter 9). DAST can be included in a continuous integration/continuous delivery (CI/CD) pipeline at the following stages:

- During the test after a build, it functions as a dynamic security scanner to detect security defects
- During production it functions as a vulnerability scanner

2.2. Applying Security Testing

Security test design can be based on the following sources:

- Regulations, standards, and laws (e.g., a new regulation requiring that cars have a security test before approval)
- A completed risk analysis
- Available threat models
- An ad-hoc categorization of security risks (see [ISTQB_ATTA_SYL])
- A security test approach
- Security requirements of security functions and mechanisms
- Systems and products in the software development lifecycle (SDLC)
- The STE's experience and testing skills
- Previous incidents where security was (almost) breached
- A reference test guide, such as the [OWASP Test Guide]

The following are attributes of a security test that should be considered during security test design:

- Required (mandatory) regulations or laws
- Identified security risks and threat models prioritized by the security test approach
- Traced to defined security requirements
- Defined according to the intended testers (e.g., developers, functional testers, and STEs)
- Defined according to security defect profiles and known vulnerabilities
- Designed to be automated, if applicable
- Destructive testing or non-destructive testing
- Intrusive or not intrusive (e.g. is the objective to break a system from within or to bring down a system via a distributed denial of service)

The basic workflow of security test design is:

1. Analysis of the security test approach (at the project level)
2. Analysis of security risks, threat models and requirements (at the project level)
3. Application of security test techniques

In most cases, an efficient security test design is based on a mix of the above sources. Depending on the type of project, it is important to ensure that a security test is performed in every phase of the SUT's SDLC.

2.2.1. Addressing Security Risks in Test Design

A key principle is that the STE should be able to create and implement security test cases based on any identified security risk, security requirement, threat and experience.

Security testing can be based on external security risks in the production environment (threats on a system or product), a security testing approach, and other sources such as threat models. Security risks can also be seen as functional and structural in nature (i.e., risks due to lack of security by design).

During security test case design, the STE must identify whether a test is destructive or non-destructive. If a test is identified as destructive it must ensure that it does not have any negative impact on other test activities, environments or the business.

In the following, common security risks and vulnerabilities at the functional and structural levels are described, along with appropriate security test techniques.

Functional Security Controls, Risks or Vulnerabilities

Tests are designed to verify and validate that controls are in place, that they work correctly, and are effective in detecting and preventing unauthorized actions.

Security test techniques are based on requirements for functional security controls and functional access controls.

Structural Access Controls, Risks or Vulnerabilities

Tests for these controls are based on how user rights have been established for data access, functional access and privacy levels. Structural access controls are typically applied by a system administrator, security administrator or database administrator. In some cases, access rights are a configuration option in an application. In other cases, access rights are applied at a system infrastructure level.

Operating System Access Risks or Vulnerabilities

Once access is gained to the operating system, an attacker can control processes, data and network access, which may enable malware to be inserted.

Platform Risks or Vulnerabilities

Each platform has its own set of security vulnerabilities.

Security test techniques are based on the STE’s experience (e.g., in dealing with vulnerabilities) and testing of security procedures (e.g., testing maintenance of security conditions).

External and Internal Threats

Some threats, such as exploiting application or programming language vulnerabilities, can be detected, tested and their impact limited. Internal threats are executed by internal employees. External threats are executed by external people (e.g. attackers).

Security test techniques are based on exploratory testing (e.g., to find potential targets and useful injection points/attack vectors) and the STE’s experience.

2.2.2. Reconciliation Testing and Recertification Testing

Understanding Identity and Access Management Concerns

The business services provided by organizations are increasing in complexity. They are deployed as a system of systems and hosted in hybrid environments consisting of elements which are in-house, supplied by partners, supplied by customers, and in the cloud. In this distributed environment, the security management of user accounts and user rights is critical. For example:

- The user must have the right privileges, and no more
- Rights must be revoked after an employee has left the organization
- Rights management must be compliant with regulations e.g., General Data Protection Regulation (GDPR)

Identity and access management (IAM) is a discipline to manage and maintain user accounts and rights by defining who (identity) is willing to access what (role) for a specific resource. IAM lists two sub-processes, which are directly related to security testing:

- Reconciliation:
  The comparison and updating of user access, rights and privileged accounts via change requests and approving chains to an authoritative trusted identity management database.
- Recertification:
  Regular reviews of accounts and related rights and privileges to verify if they are still required.

Reconciliation is necessary to ensure that all application accesses are synchronized with the same “trusted source”. Levels within the reconciliation process are:

- Full: Comparison of all accounts and user access attributes with the identity and access management system, with the purpose of identifying any differences
- Incremental: Compare only the changes to accounts and rights created, updated, or deleted
- Automatic: Where applications are used which can schedule automatic comparisons of any security relevant changes made

Recertification consists of auditing a user account and access privileges to determine if they are still justified, consistent with the organization’s internal policies, and compliant with regulations. This involves a continuous audit being performed to ensure that users only have access to what they need and what they have permission for. The assessment could be:

- Manual
  - Extract and collate accounts information
  - Present information
  - Review by managers
- Automated
  - Messages are sent to managers to issue recertification requests.
  - This has the advantage of being able to plan audits on a regular basis.

Within a large organization, with systems hosted on a wide range of environments, IAM becomes complex because of the need to manage several applications, each with accesses to be granted and revoked according to a user's movements (e.g., arrival, leave, and transfer). In some organizations, user accounts and privileges are managed outside of a formal IAM process to save time. This is a critical issue from a security point of view because orphan and unused accounts may result in exploitation by an attacker.

How to Perform Reconciliation Tests and Recertification Tests for Identities and Permissions Mechanisms

Reconciliation tests and recertification tests must be performed to avoid inconsistencies in user accounts across all the applications which the user has access to. This includes aspects such as login credentials and privileges.

The following test conditions apply:

- New account management
- Modification of account credentials and privileges
- Relocation of an account, including removal of application access and adding access to new applications
- Review of all accounts

Test objectives can be:

- Attempting to switch, change or access another role
- Reviewing the granularity of the roles and the needs behind the permissions given
- Verifying that the identity requirements for user registration are aligned with business and security requirements.

Security test techniques for reconciliation and recertification include:

- Reviewing documentation for the reconciliation and recertification processes
- Checking if registrations have been vetted by a person prior to provisioning, or whether they are automatically granted when particular criteria have been met
- Verifying, vetting and authorizing of de-provisioning requests
- Checking that account modifications are effective
- Performing fuzz testing on possible roles to be sure that the system rejects fuzzed roles
- Reviewing role permissions after gathering all applied modifications

Note that the [OWASP Test Guide] provides a list of security test objectives and test techniques related to IAM testing.

2.2.3. Testing Identification, Authentication and Authorization

Understanding the Relationship Between Authentication and Authorization

The sensitive assets of an organization must be protected and must only be accessible to an authorized person who has been previously authenticated. Identification is the first step of gaining access to a resource. It is the process of asserting an identity.

Authentication is based on the verification of a user identifier and a token to answer the questions:

- Who is the user? (user identifier)
- Is the user really who they allege to be? (token, such as a password or certificate)

Different implementations of authentication mechanisms might be used depending on the level of protection to be given against attacks to hijack an authentication or to steal the token.

Authorization is used for the following purposes:

- To verify if the authenticated user has the rights to perform an action
- To determine which level of access should be allowed to the system resources

There is a strong relation between authentication and authorization based on the principle that a non-authenticated user must not have privileges or have restricted privileges on the system.

The subject's authentication, authorization and accounting give rise to the abbreviation AAA, which is a framework which helps to control and track access within a computer network. Accounting is the third "A" in the AAA framework which considers the logging, tracking of and activities of a user.

How to Test Authentication and Authorization Mechanisms

Security test techniques (penetration tests) for authentication mechanisms could include:

- Testing for default credentials
- Testing for weak password policy
- Searching for leaked information using open source intelligence (OSINT)
- Brute force tests using dictionary and rainbow tables (i.e., precomputed tables of reversed password hashes) to perform attacks which attempt to discover user passwords. The first steps might be, for example, to try "123456", "111111", date of birth, or the name of a pet.

Security test techniques (penetration tests) for authorization mechanisms could include:

- Exploiting a lack of input filtering, such as injecting SQL requests to be authenticated without any known login/password or causing input buffer overflow to get administration access to a shell session.
- Entering an unauthorized Uniform Resource Identifier (i.e., ../../ in an File Transfer Protocol account) or a Uniform Resource Locator (URL) (i.e., site address/admin) to try to gain access to sensitive data

- Testing for horizontal and vertical privilege escalation violations

Note that the [OWASP Test Guide] provides a list of security test techniques for testing authentication and authorization.

2.2.4. Encryption

Understanding Encryption

An encryption mechanism can be used to avoid divulging sensitive data, even if it can be accessed when stored somewhere or exchanged between a client and a server. Encryption is a process of encoding data, (e.g., plain text), into cyphered data (e.g., cyphered text), using a cryptographic algorithm and secret keys. The secret is shared and only known to the authorized users. The goal is to use encryption that is strong enough to prevent an attacker, who may have succeeded in stealing encrypted data, from recovering the plain text. The use of cryptographic algorithms helps to ensure the confidentiality and the integrity of sensitive assets and to prevent them from being manipulated.

The primary and typically used cryptographic protocol types are:

- Symmetric encryption: based on the use of a shared secret key
- Asymmetric encryption: based on the use of private, public key and certificates, managed via a public key infrastructure

How to Test Common Encryption Mechanisms

Some cryptographic mechanisms are known to be weak, especially due to the short length of the secret keys. Other mechanisms might be vulnerable because they are either not implemented using best practices, or they embed coding defects (e.g., buffer overflow).

Tests of encryption mechanisms should include:

- Conformance tests (e.g., security requirements-based tests) of encryption mechanism implementations
- Tests for “by design” vulnerabilities
- Tests for “by construction” vulnerabilities
- Tests for “by configuration” vulnerabilities

Note that [OWASP Test Guide] provides a list of security tests for checking weak cryptographic implementations.

2.2.5. Testing Protective Technologies

STEs need to understand the nuances of different lines of defense so that appropriate tests can be designed to verify and validate their effectiveness.

How to Test System Hardening

Testing the effectiveness of system hardening can be accomplished in a variety of ways. System hardening restricts the access of the system to the right roles, opens only the needed services, and monitors application updates. Therefore, to test the effectiveness of system hardening, tests should be designed to detect whether the system hardening measures are working, applied in the right places, and in the right ways. It is also relevant to test for system hardening protections that are too restrictive and might be excessive compared with the security risks.

Some system hardening tests may be based on a review or an audit, while others may be based on the ability of certain user groups to perform certain actions, or to access certain data.

How to Test Firewalls

Due to the number of protocols, their different options and the complexity of the networks to be protected, it is difficult to configure a firewall efficiently and consistently. Tests for firewall effectiveness should include:

- Performing an audit to check the firewall configuration
- Port scanning to verify if the security policy is well implemented
- Using malformed network packets and network fuzz testing to exploit unexpected behavior
- Fragmentation attacks to bypass filtering features with the objective of carrying out an attack behind the firewall
- Targeting the web application firewall by encoding and compressing data or obfuscating it to hide the malicious information that represents the attack. Web application firewall evaluation criteria [WAFEC] can be used to test the effectiveness of a web application firewall.

How to Perform Intrusion Detection

Scenario-based detection is based on a known scenario or “signature”. It is easy to bypass because only known attacks are detected. Tests could include the following evasion techniques:

- Character encoding or modification of data (e.g., adding white space and end of line indicators)
- Internet protocol (IP) fragmentation, transmission control protocol (TCP) segmentation
- Encryption or obfuscation
- URL encoding

Behavior-based detection is based on a model of the system behavior and generates a large number of false-positive results and false-negative results. A false-negative result is any security alert that should have been reported but was not. False-negative results can occur when a new attack is developed that an intrusion detection system (IDS) is not aware of, or perhaps a rule might be written in such a way as to detect some attacks but miss those not specified in the model.

The accuracy of this detection method should be maintained. It is possible for an attacker to deviate from normal IDS behavior, which results in a new specification for intrusive behavior. Complementary tests should use malicious traffic to add new intrusive specifications to be considered as authorized traffic.

How to Perform Malware Scanning

Developers of malware use different techniques to protect their code against reverse-engineering and detection by anti-malware software. Some of these techniques include:

- Exploiting system library functions used by anti-malware
- String obfuscation to disable the understanding of malicious code behavior
- Code permutation
- Insertion of unused code
- Dynamic loading of functions and libraries (e.g., to limit the analysis of the malicious code)
- Automatic update of applications

From the functional suitability testing perspective, signature based anti-malware tools could be used to test the effectiveness of anti-malware without developing real malicious pieces of code. Other types of malicious files must be tested regarding the type of applications

The testing of behavior-based anti-malware is difficult because there is no clear understanding and definition of what malicious behavior is. Ideas for testing can benefit from techniques used by developers of malware:

- Unsigned execution files trying to use system calls to make system changes
- Trying to launch unusual processes with granted rights
- Attempting to copy execution files in unauthorized locations
- Trying to call unusual system APIs

An important consideration when implementing new anti-malware (either based on signature or behavior) or upgrading existing anti-malware is to test the implementation on a representative platform before deploying it to the entire organization.

Testing Data Obfuscation

Strict configuration control between the obfuscated data and keys used for the obfuscation is needed to ensure the correct versions of keys are used. Otherwise, the data cannot be understandable for use.

Since private data could be involved in some tests, data obfuscation may be used for testing purposes to render production data used in a system test environment anonymously. Sensitive data, such as user information used by a health information system, must not be divulged to testers. Tests could include brute force or dictionary attacks to attempt to get plain data from obfuscated data.

Tests to verify obfuscation of code could include:

- Reverse-engineering of code
- Brute force attacks, because some obfuscation mechanisms are vulnerable

3. The Security Test Process - 120 minutes (K3)

Keywords

risk, test environment, test plan, test process

Security Keywords

None

Learning Objectives for Chapter 3:

3.1 The Security Test Process

STE-3.1.1 (K2) Explain different activities, tasks, and responsibilities within a security test process

STE-3.1.2 (K2) Understand the key elements and characteristics of an effective security test environment

3.2 Designing Security Tests for Test Levels

STE-3.2.1 (K2) Give examples for security tests on the component test level based on a given code base

STE-3.2.2 (K2) Give examples for security tests on the component integration level based on a given design specification

STE-3.2.3 (K3) Implement an end-to-end security test which validates one or more security requirements related to one or more business processes

3.1. The Security Test Process

Security testing is a process within an SDLC. The security test process is dedicated to the security scope and must be aligned with the development process so that appropriate test activities are performed when needed.

Each organization's security risks and needs are unique due to the nature of the organization, the technical environments, the SDLC and the business risks. Therefore, the security test process must be defined and implemented in the context of these factors.

3.1.1. ISTQB Security Test Process

Table 3.1 shows how to take into account and handle security test activities within the ISTQB Fundamental Test Process.

Table 3.1 – ISTQB Security Test Process

ISTQB Security Test Process Activity	Security Test Tasks	Responsibilities
Security test planning: The goal is to define an appropriate scope of security testing that corresponds to the security risks.	• Take requirements related to security into account • Define security test objectives • Define the scope of security testing • Identify security test resources • Define test estimates and schedules for security testing • Define security test metrics, entry criteria and exit criteria	A security test analyst (STA) is responsible for this task. The STE contributes to the planning by estimating test workload and necessary hardware and software resources

Security test analysis:
The goal is to gain understanding of all security test conditions and determine what to test.
	• Review the test basis for security testing, such as security risk assessments, requirements related to security, security-based architecture and security policies
	• Define security test conditions based on:
	  • Security objectives
	  • Security risks
	  • Security standards and known vulnerabilities
	  • Defences implemented to secure the system and its data
	  • Scope of security testing
		An STA defines the security testing scope

Security test design: The goal is to identify high level test cases, i.e., how to test	• Design security test cases and test suites • Prioritize test cases and test suites • Identify necessary test data either for vulnerability assessment or penetration testing • Design security test environment (i.e., infrastructure and tools) • Set traceability between the test basis and test cases	The STE designs and prioritizes security test cases An STA reviews the STE's work products
Security test implementation:	• Organize security test cases into test procedures or test scripts. Set	An STE implements the security test cases

ISTQB Security Test Process Activity	Security Test Tasks	Responsibilities
	up a test environment to perform security testing	
Security test execution:	• Perform functional suitability security tests • Perform penetration tests • Determine specific vulnerabilities • Report with detailed information the interim security test results to management	The STE executes the security tests, produces detailed test results and communicates identified vulnerabilities as soon as possible
Security test monitoring and test control:	• Monitor security test progress and test results • Take corrective actions as needed in response to information gathered	The STA is responsible for this task.
Security test completion:	• Ensure all planned security tests have been performed • Analyse security test results to evaluate residual risks • Analyse security test results to improve software development in terms of security • Report the final security test results to management and other authorized stakeholders • Determine if security testing deliverables (i.e., test reports) have been delivered • Archive test results, test data, and other sensitive information in secure locations	The STA collects all information produced during security test execution and produces a high-level test report to management

Where exploratory testing has been performed, the security test design, security test implementation and security test execution are based on test results from previous tests using standard techniques for exploring such as sniffers, scanners, brute force attacks, and bots. The test design, test implementation and test execution are continuously achieved.

3.1.2. The Security Test Environment

While many types of tests can use a test environment located on the same server(s) and network(s) with other systems, security testing has specific risks, even if it is virtualized or container-based. For example, performing destructive security tests, contamination of the SUT and corruption or divulgation of data require a segregated approach to building a test environment for security testing. Moreover, in most business domains, regulations require that different environments be used for development, testing and production. For example, the payment card industry data security standard (PCI DSS) Requirement 6.4.2 states that separation of duties between development, test and production environments is required. Similarly, PCI DSS Requirement 6.4.3 states that production data (i.e., personal area networks) shall not be used for test or development [see chapter 6, PCI DSS].

The security test environment must contain all needed functions with which to perform the tests. These include test management tools, security test tools, and test automation tools. These are needed to enable the discovery of as many vulnerabilities as possible within the allocated time, and with as few false-positive results and false-negative results as possible.

It could therefore be necessary to identify, specify and set up an effective separate security test environment to protect other environments such as development, component testing, component integration testing, system testing, acceptance testing, and production. This effectiveness must cover either the fault tolerance regarding destructive security tests or provide protection against other systems under test and for the productiveness of security tests.

The STE must analyze and estimate the required architecture, the APIs and the behavior of the SUT to appreciate the impact of security testing and define the most effective test environment.

The main characteristics of a security test environment include:

1. Isolated at the right level (if necessary):
   Depending on the risks, the SUT is isolated either via filtered communications, or the SUT and all other dependant systems are isolated from other environments (e.g., a merchant website needs a separate payment management service).
2. Target environment representative:
   To obtain the correct behaviour of the SUT, the total environment must reflect the production environment in terms of exact version and configuration.
3. Productive:
   Contains all tools needed to plan, prepare, execute and report security tests, either in a manual or (where possible) automated manner. Security test execution needs specific test tools, as described in chapter 9.
4. Recoverable:
   To repeat tests as needed and to recover from corruption should it occur

3.2. Designing Security Tests for Test Levels

Threat modeling is a repeated activity in which each security test level must be adjusted regarding the test results of the latest threat modeling results.

Depending on the type of project, it is important to ensure there is a security test planned in every applicable SDLC phase.

3.2.1. Security Test Design at Component Test Level

Test Basis for Security Test Design at the Component Test Tevel

Examples of work products that can be used to design security tests include:

- Risk analysis
- Requirements of security functions and mechanisms
- Detailed design of security functions and mechanisms (e.g., APIs and algorithms)
- Data models
- Compilation or building rules
- Compiler information

Test Objects for Security Test Design at the Component Test Level

Typical test objects for security component test include:

- Components
- Dependencies (e.g., third party libraries)
- Source code
- Database modules

Typical Security Defects and Failures at the Component Test Level

Examples of typical security defects and failures that can be found at the component test level include:

- Incorrect code and logic
- Incorrect behavior
- Input filtering weaknesses
- Data flow problems
- Call flow problems
- Unreachable (dead) code
- Deliberately inserted malevolent code (i.e., software bombs)

Types of Security Tests at the Component Test Level

Static tests and dynamic tests can be applied at the component test level.

Depending on the security test objectives, the test basis, test objects and test types, different design approaches and techniques can be used at the component test level based on the given code:

- Verifying that implementation of security functions and mechanisms behave as expected by security requirements

  The design of security tests is based on detailed requirements (e.g., detailed specifications and detailed design of the SUT). Well-known test techniques based on specifications should be used, like boundary value analysis and equivalence partitioning [ISTQB FL]. The STE must trace the security test cases to the detailed specifications.

- Building confidence in the quality of security code (i.e., secured coding)

  Security test cases must be focused on the application of secure coding rules. The STE must also verify that the development team does not use dangerous code instructions and avoids weaknesses in programming languages and compilers. Usually, development teams or organizations define their own secure coding best practices based on well-known references, which may be internal to the organization, or external like the OWASP foundation. The STE can design test cases based on these rules that can be considered as non-functional requirements (e.g., maintainability and other non-functional quality characteristics). These security test cases can be processed automatically using static analysis tools.

  Tests for any component should include assessment of possible violations of the following checklist of best practices: [CERT1]

  - Validate inputs
  - Heed compiler warnings
  - Architect and design for security policies
  - “Keep it simple” principle
  - Default denial, which defines an “allow” list
  - Adherence to the principle of least privilege
  - Sanitize data sent to other systems
  - Practice defense in depth
  - Use effective quality control techniques
  - Adopt a secure coding standard

Tests performed against such best-practice checklists should include assessments of possible violations of these practices based on a well-documented risk analysis incorporating realistic threat modelling.

- Finding vulnerabilities in the components

After verifying the correct implementation of security functions and mechanisms and that secure coding best practices have been followed, the STE should design security tests with the objective to find vulnerabilities in the developed components (e.g., fuzz testing the API of a component).

The STE can use static application security testing (SAST) or dynamic application security testing (DAST) tools to help find vulnerabilities.

- Mitigating security risks

All security tests described above help mitigate the security risks of the developed application or system.

Analysis of Security Test Design at the Component Test Level

One key measure of adequacy of security test design involves evaluating coverage. Various coverage measures come from the tests performed.

Coverage may be measured as any of the following:

- Percentage of total number of security requirements tested
- Percentage of specified use/abuse security cases tested
- Percentage of critical security functions, scenarios, or mission threads tested
- Percentage of source code coverage (e.g., to identify dead code or backdoors)
- Percentage of data equivalence partition coverage (e.g., to detect bad exception catches)
- Number of security findings
- Efficiency of security tools used (e.g., number of false-positive results and false-negative results)

3.2.2. Security Test Design at the Component Integration Level

According to ISTQB Foundation Syllabus [ISTQB FL], there are two different levels of integration: component integration testing and system integration testing. Components and/or subsystems to be integrated can come from a variety of different sources, such as another team in the same organization, a subcontractor, a commercial off-the-shelf (COTS) product, a service already available in the cloud or an open-source service. During these integration activities to ultimately build the full production system, the possibilities for security breaches are not simply the summation of the vulnerabilities in each of the components. Instead, new attack vectors become possible due to interactions between components within the larger system and because of organizational elements.

However, some interactions between components might mitigate or block possible sequences leading to security breaches.

Component integration testing can demonstrate the complexity of a system design and the stability of its behavior. The component integration test approach (e.g., top-down or bottom-up) can affect the timing of revealing security concerns or the need for additional security-specific tests

As with component tests, component integration tests should be designed on the basis of well-documented risk analysis incorporating realistic threat modelling. As separate components are integrated together, note that scaffolding or mocking in the form of stubs and drivers may be necessary to test incomplete paths through a system during integration. As more implemented components are added to the system, scaffolding/mocking is incrementally removed, allowing for fuller assessment of functional suitability as well as opening up new paths to vulnerabilities that might be exploited.

According to the level of trust in the components / subsystems to be integrated, the security test design at the component integration level should include:

- Security tests of the global security architecture based on technical architecture documentation
- Security tests of configured integrated flows (e.g., authorized or not and the level of confidentiality, integrity and availability)

- Security tests of integrated APIs (e.g., to detect security issues at APIs due to a lack of controls or a lack of knowledge of these APIs)
- Security tests regarding the security configuration of integrated components (e.g., filtering access of a component by another since unsigned components should have limited access)
- Verification that integrated components which are external, open-source or closed source software are free of vulnerabilities

At the component integration level, coverage may be measured as any of the following:

- Percentage of used/tested APIs
- Percentage of tested interactions between components / subsystems based on technical architecture documentation
- Number of security findings at the component integration level
- Number of security findings that should have been found at the component test level
- Efficiency of security tools used (e.g., number of false-positive results and false-negative results)

3.2.3. Security Testing in System Testing and Acceptance Testing

Security System Test

This is the test level during which the implementation of security requirements is tested to ensure that they function as expected. System security testing activities include performing security tests in some approximation of the production environment, necessitating that a transition takes place away from the development environment in which the preceding implementation and integration activities have occurred.

The Role of Security Testing in System Testing

Security system testing is the first opportunity for exercising the end-to-end functionality of the fully integrated components. Although usually done in a test environment, it should reveal emergent properties of the system that would not have been observed before integration is completed. Security requirements are typically considered in conjunction with the functional requirements.

The objective of security testing in system testing is to test:

- all the security requirements implemented in the security functions in a test environment representing the production environment
- that the operational configuration is secured

Security Acceptance Testing

This is the final level of testing during which users, or their representatives, build confidence that the system is able to deliver the necessary capabilities in the production environment in a secure manner. The security acceptance test objectives include security testing against the security-related acceptance criteria established for the system. Typically, the security-related acceptance criteria focus on functional security controls and processes. The security acceptance test activities may include:

- Installing the system into a pre-production environment
- Performing security tests based on acceptance criteria
- Determining acceptance based on security test results

The Role of Security Testing in Acceptance Testing

Acceptance testing is distinguished from system testing in that it is performed in an environment resembling production. It finally places the system in the setting where external threat agents would be seeking to find weaknesses on a day-to-day basis. These tests allow for reasonable evaluation of performance efficiency and other behaviors based on interactions through external interfaces.

Acceptance testing should ideally validate that the initial project goals have been delivered and that documented security acceptance criteria are met. This is accomplished by designing and performing tests to validate security processes / scenarios such as rights control, authorization management and firewall filtering.

The best time to define and document acceptance criteria is before system development or purchase. In the context of security testing, the acceptance criteria may be global in nature. For example, there could be acceptance criteria that specify what is acceptable in terms of overall system security. This would include criteria that are applied to all system functions, such as user authentication, user rights, encryption levels and audit trails.

Analysis of Security Test Design at the Acceptance Test Level

At the acceptance test level, coverage may be measured as the following:

- Percentage of tested security processes / scenarios
- Number of security findings that should have been found in previous test levels with their severity
- Efficiency of security tools used (e.g., number of false-positive results and false-negative results).
- Percentage of tested security requirements
- Percentage of tested operational secure configuration items

4. Security Testing Standards and Best Practices - 195 minutes (K3)

Keywords

Common Attack Pattern Enumeration and Classification (CAPEC), Common Vulnerabilities and Exposures (CVE), Common Vulnerability Scoring System (CVSS), Common Weakness Enumeration (CWE), Common Weakness Scoring System (CWSS), vulnerability, weakness

Security Keywords

None

Learning Objectives for Chapter 4:

4.1 Introduction to Security Standards and Best Practices

STE-4.1.1 (K3) Explain different sources of standards and best practices and their applicability

4.2 Apply Important Standards and Best Practices for Security Testing

STE-4.2.1 (K3) Apply the concept of the Open Web Application Security Project, Common Vulnerability Enumeration, Common Weakness Enumeration, the Common Vulnerability Scoring System and the Common Weakness Scoring System and how to leverage them for security testing

4.3 Leveraging Security Testing Standards and Best Practices

STE-4.3.1 (K2) Explain the advantages and disadvantages of test oracles used for security testing

STE-4.3.2 (K3) Understand the advantages and disadvantages of using security best standards and best practices

4.1. Introduction to Standards and Best Practices

Standards and best practices of various types provide visibility into professional consensus and regulatory obligations. A consensus-based standard represents the considered opinion of a knowledgeable body of experts.

Even if often used as synonyms, there are big differences between standards and best practices, which are explained in the following subsections. The differences have a significant impact on the selection process and possible use cases for utilizing them.

4.1.1. Standards and Best Practices

Standards

Standards are defined as “a document, established by a consensus of subject matter experts and approved by a recognized body that provides guidance on the design, use or performance of materials, products, processes, services, systems or persons.” ([ISO_Web_21], and Appendix D).

There are several levels of a “recognized body”, which allows for a distinction between different types of standards. One of the highest levels of recognition of standards is represented worldwide by the International Standard Organization (ISO). Usually, each country that is part of the World Trade Organization (WTO), has its own local representation. Standards have the highest level of recognition for an STE because of their high level of maturity. However, this maturity has the disadvantage of taking a lot of time to complete and often results in standards with a very reactive character.

Recognized bodies may create their own standards. These can be categorized as industry standards, de facto standards and manufacturer specific standards:

- Industry standards:
  These have been established over years of application in many contexts and have demonstrated some added values by solving a particular problem. The Internet Engineering Task Force (IETF) is an important player in creating standards at this level. They make standards based on the combined engineering judgement of their participants and their real-world experience in implementing and deploying their specification [IETF23].
- De facto standards:
  These often have their roots in industry standards. Since their coverage and acceptance is high, they even fulfill many of the criteria to be considered at the highest level of standards. A good example is the TCP protocol, that was established as an industry standard but is today considered to be a de facto standard [IETF23].
- Manufacturer-specific standards:
  Some clients/organizations have learned that there is added value in following the proprietary specifications of a specific manufacturer.

In real life this clear classification might have many fuzzy overlaps, and it is not always simple to do a clear classification of a given standard.

Best Practices

Best practices and their recognized body are less formally organized. Gartner’s Glossary [Gart21], defines best practices as a “group of tasks that optimizes the efficiency (cost and risk) or effectiveness (service level) of the business discipline or process to which it contributes. It must be implementable, replicable, transferable and adaptable across industries”. At this level, every group of experts, even if they are working within the same context, can create their own set of best practices.

4.2. Apply Important Standards and Best Practices for Security Testing

Several standards and best practices exist for the discipline of security testing. Due to the high level of requirements to be fulfilled to be considered a standard, their creation and maintenance is much slower than for best practices. This allows for deep recognition within industry and includes many feedback loops for improvement. However, this impedes the quick adjustment to new trends and risks. In comparison, best practices have a high overall performance efficiency, but it is more difficult for them to become well known, to achieve a high level of coverage, and to be empowered by practical evidence.

4.2.1. Industry Standards for Security Testing

The most established standard in IT security is the series of ISO 27000. The [ISO 27001] standard is internationally accepted and entitled “Information technology — Security techniques — Information security management systems — Overview and vocabulary”. The focus of this standard is on information security management, i.e., to identify risks, to evaluate them and to manage them through information security controls. All these activities are combined in an information security management system (ISMS) which is the overall core of the ISO 27000 standard. The standard is broad in scope and focuses on the general way in which an organization should assess their risks, contrast them with their specific needs and deal with the most relevant risks. The core standard can be applied to all organizations.

The ISO 27000 series consists of more than 40 individual standards, which can be classified into the following:

- Main standard: General overview and introduction to an ISMS (starting from ISO 27000 to ISO 27005)
- Topic specific standards to cover specific topics like service management ISO 27013 and public cloud provider ISO 27017 (see [ITGOV23a])
- Domain specific standards to focus specific domains like telecommunication providers ISO 27011 and financial industry ISO 27015 (see [ITGOV23a])

The most used standards that apply to the context of security testing and cover the relevant test objects and test conditions which the STE should consider, are the following:

- ISO 27000: This part explains the overall structure of the ISO 27000 series and introduces an ISMS and the role security testing can play.
- ISO 27001: This is the most used standard, as it lists a comprehensive set of recommendations and security controls to structure and build an individual ISMS. Its focus is to establish a comprehensive view on the relevant assets within an organization, their exposed risks and possible mitigations. [ISO 27001]
- ISO 27001, Appendix A:
  The most important part of ISO 27001 for an STE is presented in this appendix. It lists security controls for different aspects such as access control, disaster recovery and network security. Each of these controls, if applied in a specific context, are important inputs for an STE, as it is their task to measure the effectiveness of a security control. [Cald11]

- ISO 27002: This standard takes the generic security controls from ISO 27001 and gives some more guidance on how to apply them in practice and how to specify them in more detail for a specific context. [Cald11]
- ISO 27003: This standard supports an organization to create a plan to establish an ISMS based on ISO 27001.

De facto Standards for Security Testing

There are many de facto standards which can be leveraged by an STE. One of the most important series of de facto standards stems from the MITRE corporation, even though its main business is not to generate standards. MITRE is a private, not-for-profit organization which provides engineering and technical guidance for the federal US government. The most important sponsors of MITRE are the Department of Defense, the Federal Aviation Administration and the Department of Homeland Security [MITRE21].

For the area of security testing, MITRE hosts and maintains the following well-known standards that provide added value for the STE:

Common Attack Pattern Enumeration and Classification (CAPEC™)

CAPEC™ provides a publicly available catalogue of common attack patterns. The idea is to get a better understanding of how attackers exploit weaknesses in applications and other cyber-enabled capabilities. Attack patterns are based on software design patterns for attackers. Two typical entry attack patterns are SQL injection (CAPEC-66) and relative path traversal (CAPEC-139) [CAPEC21].

CAPEC provides different views on its data sets. The most relevant ones are:

- Domain of attack, such as software, social engineering and physical security. On the highest level CAPEC lists nine domains of attack.
- Mechanisms of attack, such as inject unexpected items and manipulate system resources. On the highest level CAPEC lists six mechanisms of attack.

Each test object that an STE tests should be located within this catalogue. Often CAPEC is the starting point to get an initial overview of possible attacks that might be relevant for a given system.

The following MITRE standards are used for further refinement for effective security testing:

Common Weakness Enumeration (CWE)

CWE is a list of software/hardware weaknesses. Usually, each common attack pattern has one or more weaknesses that are usable for leveraging a CAPAC attack pattern [CWE21]. CWE uses the concept of views, the most used of which are:

- Software development, such as an API, bad coding practices and permission issues. On the highest level, CWE lists 40 software development assets.
- Hardware Design, such as memory and storage issues, core and compute issues and peripherals, on-chip fabric, and interface I/O problems. On the highest level CWE lists 12 hardware design assets.

Each common weakness is an effective starting point for an STE to test whether the underlying attack pattern can be exploited.

Open Web Application Security Project (OWASP)

It is important to realize that CWE and OWASP [OWASP21] overlap and they both list common weaknesses. OWASP is well known for publishing its OWASP Top 10 ranking.

Common Weakness Scoring System (CWSS)

The more common a weakness becomes, the more important it gets to have a prioritization scheme in place. CWSS provides a mechanism for prioritizing weaknesses in a consistent, flexible, open manner [CWSS21]. Prioritization is calculated by three sets of metrics:

- Base Finding Metric Group:
  The inherent risk of a weakness, confidence in the accuracy of the finding, and strength of controls is calculated. A typical metric is technical impact", that ranges from complete control over a system to no technical impact.
- Attack Surface Metric Group:
  This calculates the barriers that an attacker must overcome to exploit the weakness. A typical metric is required privilege, that ranges from no privileges required to administrator privileges.
- Environmental Metric Group:
  This calculates the characteristics of the weaknesses that are specific to a particular environment or operational context. A typical metric is business impact, that ranges from the business could fail to no impact.

By using specific, predefined weights, all of these metrics can be aggregated into one overall CWSS value for one specific weakness. CWSS can handle unknown metrics by default values or by defining/focusing on an individual metric subset. In addition, many metrics of the Base Finding Metric Group can automatically be calculated by a static analysis tool.

Common Vulnerability Scoring System (CVSS)

A similar prioritization mechanism to CWSS is CVSS [CVSS21], which follows a similar approach, but assumes an existing, deployed vulnerability (see CVE below). Both, CVSS and CWSS are scoring systems for computer security: CVSS is a reactive approach because vulnerabilities already exist before ranking. CWSS is a proactive approach, as you are working with software before releasing it into production. Both approaches are often used together, even if they are not fully compatible (cf. [SecJour21]).

Common Vulnerabilities and Exposures (CVE)

CVE is a database of publicly disclosed information about security issues [CVE21]. A CVE number uniquely identifies a particular vulnerability from the list. CVE helps because it provides a standardized identifier for a given vulnerability within a specific system. If a system is affected by a specific CVE, this vulnerability is a specific instance of a common weakness (CWE) that can be used to do a specific attack (CAPEC). New entries within the CVE repository usually originate from the daily work of STEs. If they identify a new vulnerability unknown to CVE, they can publish it at CVE to engage the security community to identify counter measures.

Best Practices for Security Testing

Best practices only need to achieve a low formal criterion to be considered as best practices. Many best practices may fail after some time if they don't help. Some will stop being used because of missing publication/marketing, but a few might improve their maturity on their way to being considered for a standard.

One typical mature best practice that is still used today is the STRIDE model which was invented by Microsoft [Micro09]. STRIDE allows for systematic threat modeling from an attacker perspective. The term itself is an acronym for six threat categories, which classifies potential threats: **s**poofing, **t**ampering, **r**epudiation, **i**nformation disclosure, **d**enial of service and **e**levation of privilege.

- Spoofing identity, i.e., to claim within a system to be a person or system you are not
- Tampering with data, i.e., the malicious modification of data
- Repudiation, i.e., threats that take aim at auditing and tracing, ensuring that bad behaviors cannot be proven
- Information disclosure, i.e., the exposure of information to individuals who are not supposed to have access to it
- Denial of service, i.e., to deny service to valid users
- Elevation of privilege, i.e., an unprivileged user gains privileged access.

Generally, STRIDE is used to support developers to consider threats during design and to close identified gaps. The STE can use the same approach to focus on testing.

4.3. Leveraging Security Testing Standards and Best Practices

There are many use cases possible for leveraging standards and best practices. In general, these use cases can be divided into mandatory applications and voluntary applications.

4.3.1. Mandatory Application of Standards and Best Practices

In this type of use case, standards and best practices are mandatory for another party:

- Security requirements for contracts:
  Best practices are an effective way to specify security requirements for software development, especially those that are delegated to a third party. Instead of listing all specific requirements, only the fulfillment of a specific standard is required, which implies fulfilling all security advice and requirements contained.
- Security requirements as a regulation:
  Even regulative institutions (e.g., in the banking domain) often use standards and best practices, which are easy to manage.

4.3.2. Voluntary Application of Standards and Best Practices

In this type of use case, the application of specific standards and best practices is a decision by the management to generate the following added value:

- Establishing a high level of security by reusing established security knowledge stored in existing standards and best practices
- Well-known evidence to demonstrate awareness for security
- General marketing purpose and creation of unique selling points in a competitive business area.

4.3.3. Test Oracles Extracted from Standards and Best Practices

A general use case for leveraging standards and best practices that is independent of being mandatory or voluntary is the notion of using powerful test oracles. At the application level the test oracle is usually the security requirement section in the specification. On lower levels, e.g., included libraries, underlying operating system, network traffic, standards and best practices can easily be used. Especially the more volatile type of best practices might list many known vulnerabilities for a given system and determine expected results to be used as evidence of being secure or unsecured. This is a powerful tool for STEs, as they only must define corresponding low-level test cases, execute them, and then compare the test results with the ones listed in the best practice.

4.3.4. Advantages and Disadvantages of Leveraging Security Testing Standards and Best Practices

Leveraging standards and best practices for security testing has many advantages, but there are some negative aspects to be considered carefully for a specific context. In general, the following advantages apply to leveraging standards and best practices:

- Consistent terminology:
  In IT there are many marketing terms, synonyms and phrases without a clear definition or distinction between them. Standards and sometimes even best practices can support the clarification of terminology.
- Reusing expert knowledge:
  Defining standards and best practices can be a time-consuming task which is usually done by security experts. Their knowledge can be captured and reused in standards and best practices.
- Benchmark and completeness double-check:
  If an enterprise uses its own specific security test framework, existing standards and best practices can be used as a benchmark to check for the completeness of their solutions.
- Improved commitment between supplier and client:
  The more established and recognized a standard or best practice is, the more efficiently it can be used as the basis for commitment between consumer (what they want to have) and supplier (what they must do).

- Easy communication about the achieved level of security:
  If an organization uses its own set of security tests without any external reference, it might be difficult to demonstrate their effectiveness. Using standards and best practices helps achieve an overall positive attitude and simplifies communication dramatically.

However, some negative aspects are possible when using standards and best practices for security testing:

- Wrong selection:
  There are many available standards and best practices, each having their own focus and necessary preconditions to be applicable. Leveraging the wrong source reduces the impact of achieving better security and might even decrease the resources available to be spent on security.
- Best practices within a wrong specific context:
  Whereas most standards have achieved high quality due to their long creation processes and long feedback cycles, best practices may come and go on a short-term basis. Especially when initially proposed, their correctness and added value is not always clear and might not have a strong link to the specific context. The use of a new, proprietary best practice which has not yet shown any evidence of creating any added value might even decrease the level of security.
- Missing customization:
  Frequently, standards and best practices define certain parameters that have to be fulfilled to be applicable in a specific context. If this is omitted or not done properly their application might yield limited added value.
- Commodity considerations:
  The more established and popular a standard or best practice becomes, the less it can be used to create uniqueness compared to other competitive products (if required).
- Operational Blindness:
  Flaws or the rise of new threads could lead to lower attention if standards or best practices are not adopted in a timely manner.

5. Adjusting Security Testing to the Organizational Context - 195 minutes (K4)

Keywords

weakness

Security Keywords

rootkit

Learning Objectives for Chapter 5:

5.1 The Impact of Organizational Structures in the Context of Security Test

STE-5.1.1 (K3) Analyze a given organizational context and determine which specific aspects to consider for security testing

5.2 The Impact of Regulations on Security Policies and How to Test Them

STE-5.2.1 (K3) Analyze the impact of regulations on security policies and how to test them

5.3 Analyze an Attack Scenario

STE-5.3.1 (K4) Analyze an attack scenario and identify possible sources and motivation of the attack

5.1. The Impact of Organizational Structures in the Context of Security Testing

Information security cannot be achieved by only securing the infrastructure and relying on technologically implemented measures. The people and processes of an organization must be considered as well.

5.1.1. Analyze a given organizational context and determine which specific aspects to consider for security testing

People often become a victim of social engineering attacks and important processes, such as a defined emergency response, can be either missing or improperly implemented. An STE therefore needs to cover these aspects during security testing, because they both affect the information security of an organization. People have a defined role. Depending on their role, they are involved in different processes. Roles and processes are usually strongly dependent on the organizational structure.

Organizational structure can be broadly classified into the following three types:

- Functional structure: organized by common functions, such as production, marketing, human resources, IT and accounting
- Divisional structure: Organized as a collection of functions which produce a product
- Matrix structure: employees are grouped by both function and product simultaneously

The way in which these organizational structures impact information flow and the implementation of administrative decisions is considered below:

- In a functionally structured organization, information needs to be exchanged between the different departments across the organization and administrative decisions are implemented directly in a top-down approach from the management to the whole organization
- Divisionally structured organizations add an administrative layer at the top of each division, and therefore decisions can affect only a single division. In addition, the information flow between divisions is slightly reduced, even though the divisions often contain similar departments, such as development
- Matrix structures try to unite both aspects. They are functionally separated but also adopt a product-based focus without adding the separate administrative layer found in divisional structures. However, there is a higher risk of conflicts since decisions can be made from both a product perspective and from a purely administrative perspective.

Bringing this to the context of security testing, an STE should be aware of the organizational structure for the following reasons:

1. The test results of a security test will be influenced by the department that has ordered and planned the test.
2. Depending on the overall structure of the organization, an STE can take advantage of weaknesses in the information flow.

The first of these two reasons results from the fact that IT and security departments are usually authorized to implement and assure security, but other departments may also have awareness of that. For example, knowing that an employee of the penetration testing team is visiting another department will increase the chance that people in that department actually keep to the security policies, at least for the time of their stay.

The second of the above two reasons results from the likelihood that for each organizational structure, particular weaknesses might exist. In a functional structure, an STE could take advantage of this as follows:

- Employees from different departments might not know people from other departments, particularly those responsible for IT administration.
- Security awareness and the acceptance of certain security measures might be significantly lower in departments where only non-technical staff are working.
- Information about an incident might reside within a single department for a period of time, resulting, in the worst case, in delays to the reaction time.

In divisionally structured organizations, similar potential weakness can be considered, although these may be transferred to divisions:

- Employees from different departments might not know people from other departments. Depending on how the IT infrastructure of the organization is maintained, they might not know who is responsible for IT administration.
- Information about an incident might reside within a single department for a period of time, resulting in delays to the reaction time.
- Depending on the size of a single division, teams might also be subdivided by functionality into smaller departments inside the division, making the point above regarding security awareness and the acceptance of certain security measures also valid for divisional structures.

Even though organizations with a divisional structure distribute similar functionalities over different divisions, some services like IT administration often reside in a central department.

For matrix organizations, a person might take advantage of possible conflicts between administrative management and product management. However, this is not always the case and, as already mentioned for divisional structured organizations, some services might be centralized.

Some considerations

The above information takes a high-level perspective of the organizational structures and their potential weaknesses. In practice, many organizations do not have purely functional, divisional or matrix structures. In particular the security management function is often handled by a central department which dictates security measures, such as an organization-wide security policy.

A security policy can be defined as “A high-level document describing the principles, approach and major objectives of the organization regarding security.” [ISTQB Glossary]. A security service according to NIST is defined as “a capability that supports one, or many, of the security goals. Examples of security services are key management, access control, and authentication.” [NIST02].

Studying the organization’s security policies can reveal potential attack vectors by making known the constraints placed on the behavior of its members as well as the constraints imposed on adversaries by security mechanisms. Company security policies may not be publicly available. Some organizational policies may only be accessible to employees or even just certain members of the staff.

An important aspect to consider in the organizational context is the way in which the organization outsources parts of their production or services. The relevant partner(s) should also be considered as potential targets for a security test. This depends on the nature and content of the contract between the two organizations that define the legal obligations. External partners are often given (limited) virtual private network (VPN) access, work on the same code repository or have an access token for a local office. Even though they often have restricted accounts, it might be the first step towards a successful attack. Another focus regarding partners in the context of security testing is the analysis of the supply chain, as attacks in this area can have serious consequences (e.g., [WIRED21]).

The general aspects covered in this section could hold for almost any organization, but neglect specific industrial issues, such as the type of product or service that an organization offers, as well as the industrial sector the organization operates in. Offering a web service for music might have different security requirements compared with a medical device used in a hospital. Precisely for that reason, regulations exist for certain sectors which prescribe requirements for processes, safety, security measures, or other domain-specific aspects. (see section 5.2). This might affect organizations developing their own products far more than organizations who, for example, sell COTS products.

5.2. The Impact of Regulations on Security Policies and How to Test Them

Security regulations drive the content of security policies which drives the information security control framework for security testing. The STA develops this control framework. With knowledge of the framework, an STE then develops and uses test cases to test the controls.

5.2.1. The Impact of Governmental Regulations on Security Regulations

Due to the strong interconnectivity of most industrial sectors, cyber security attacks can have a deep impact on a single organization and, in worst case, on the overall infrastructure of a whole country. As a reaction to this, governments have defined regulations to force critical organizations to adapt their security level to at least a minimum. Non-compliant organizations may be fined or temporarily shut down until the required security measures are implemented. Organizations that are affected by regulations therefore have an incentive to improve their security measures and security policies to at least the required security level. NIST defines laws and regulations in the context of computer security as “federal government-wide and organization-specific laws, regulations, policies, guidelines, standards, and procedures mandating requirements for the management and protection of information technology resources.” Although this might be true in some cases, general regulations can be defined on a global, union or national level, such as:

- Global regulations defined by, e.g., the WTO
- Union specific regulations such as the GDPR and Network and Information Systems (NIS) Regulations defined by the European Union (EU)
- National regulations such as the Cyber Security Sharing Act in the United States of America (USA)

In addition to this, further regulations may apply to specific industrial sectors, such as:

- Health Insurance portability and accountability Act [HIPAA]
- UNECE WP.29 for automotive sector [UNECE20]

- Implementing Regulation (DVO) (EU) 2019/1583 for aviation security [BSI21]
- PCI DSS [PCI22]

The formulation of regulations is done by specific institutions such as the Cybersecurity and Infrastructure Security Agency in the USA, or the Federal Office for Information Security in Germany. In the EU, the European Network and Information Security Agency defined the NIS directive, which was set into policy in 2016. The aim of this is to increase and standardize the cybersecurity level across all member states. The same institutions often publish recommendations (e.g., [TR02021]), best practices or at least references to other publications.

This is important because regulations can be very unspecific about which actual technology to use in practice. Due to technology changing quickly over time, this would require that an ongoing adaptation of the defined laws is needed. However, there is a common understanding about state-of-the-art technology which is published, for example, by institutions such as TeleTrust [TELETRUST] or NIST and is adapted over time.

Regulations are often unspecific because they aim to cover a broad scope of industrial sectors. Keeping to a fixed set of IT security technologies could cause problems, as some technologies might become unapplicable for certain organizations.

While the use of current technology is one part of regulations, the following three cornerstones also need to be considered:

- Resources (e.g., hardware, software and state-of-the-art technology)
- Personnel
- Information security processes

Regarding personnel, there are four main aspects to consider:

- Personnel must be aware of the importance of information security and must understand that they are responsible for ensuring security
- They must have the required knowledge to implement and to apply defined security measures noting that the type and specificity might vary according to different roles. Skills and knowledge are needed about the actual defined security policies and procedures.
- They must accept and apply the defined Information security processes (see next section)
- Some personnel require special clearances

Information security processes include aspects such as:

- Defining responsibilities. In the context of regulations, this affects roles and responsibilities inside an organization, and also the institutions which are responsible for monitoring compliance and reporting incidents, such as the National Cyber Emergency Response. Often a person is nominated as the single point of contact within an organization. They have to define when and to whom they need to make reports
- How to deal with new or leaving employees, personnel who change departments, or external employees

- In the event of a security alert, who needs to be informed, who will be responsible for taking decisions and when does the organization have to report the incident to a (federal) institution
- How to deal with business partners and suppliers
- Regular reviews and reevaluations of currently defined processes and measures
- Auditing procedures, including preparations and corrections after an audit has taken place

The aspects described above are an integral part of a working ISMS. Some regulations aim to establish and preserve an essential ISMS (see chapter 8).

How to Test Security Policies

STEs have a high level of responsibility in testing security policies for organizations affected by regulations. The main reasons for this are:

1. From an organization's perspective, it is important to be compliant. Otherwise, the organization may have to pay penalties or risk their business being temporarily shut down.
2. Since incidents for regulated industry sectors can have far more serious consequences than for other sectors, testing must be done very thoroughly to ensure a high level of security.

The next step is to evaluate best practices and state-of-the-art technologies, since regulations may only refer to them. STEs need to validate that the given security measures for an SUT are still sufficient. For example, they need to check if data is encrypted, and which algorithm is used, since this might already be unsecured.

For testing purposes, the results of previously conducted audits can be considered as part of the test basis. However, findings from a previous audit which have now been implemented need to be confirmation tested. The STE cannot assume that the implementation has been correctly performed. Furthermore, as regulations and key objectives of a security policy include people and processes, test activities must also include these aspects.

Testing of personnel aspects can be performed by applying tests such as faking a social engineering attack or trying to bypass (physical) access control. These kinds of tests are dependent on the organizational department that requested the security test.

Security testing of processes includes backup and restore, and emergency response and reporting. These tests can be very elaborate and expensive, as many people might be involved during the test

5.3. Analyzing an Attack Scenario

5.3.1. Common Attack Scenarios

Security incidents vary widely in terms of the attack techniques and tools applied, the type of attacker and the motivation for performing an attack. As a result, it is difficult to give a generic description of an attack. However, certain steps are common for almost every attack. These steps can be defined as:

1. Information gathering
2. Exploitation/gaining access
3. Persisting/maintaining access
4. Clearing tracks

By way of comparison, security incident handling is defined according to NIST [NIST03] by the following steps:

1. Preparation
2. Detection and analysis
3. Containment, eradication and recovery
4. Post-incident activity

While both enumerations describe a common sequence of attack and response, factors such as motivation, resources, the skills of an attacker and the approach used have a strong impact on both the success of an attack and the consequences for the attacked party.

Classification of Attackers and their Motivation

The word “hacker” is used in this section as a synonym for an attacker. However, the term hacker generally refers to a highly technically skilled person.

Attackers can be divided into different types, depending on their technical skills and the resources are available to them:

Type of attacker	Description
script kiddies	These are people with a low level of technical knowledge, who use existing tools and scripts without fully understanding them, and who have very limited resources.
scammers	These people use simple techniques such as phishing but usually target many people, which increase their chances
private hackers	These people have a solid technical background and are interested in IT security or are very curious
professional hackers	These people have a very high technical background and are able to perform highly sophisticated attacks to earn a living

Type of attacker	Description
governments	These organizations pay complete teams for spying, hacking or sabotage and have considerably more resources available to them than single persons or small groups.

While this is a coarse-grained categorization depending on skill and resources, another important aspect is the motivation of an attacker. Script kiddies might just want to play around with something they've just found online or impress their friends, whereas professional hackers earn their money by hacking and therefore also want to have a good reputation. The following are categories of motivation:

- Personal motivations (e.g., fame, vengeance, jealousy, and curiosity)
- Political motivations (e.g., "hacktivism", war, and espionage)
- Professional motivations (e.g., money, reputation, and industrial espionage)

Depending on the level of motivation, an attack can be performed against a single entity or several entities. For example, ransomware is a malware which usually encrypts data and blocks the use of the infected system until the victim pays a certain amount of money. Since infecting more systems will increase the chance for an attacker to earn money, ransomware usually is written to infect as many systems as possible.

In contrast to this, computer worms such as Stuxnet [WIKI02], are mainly developed for the special case of infecting supervisory control and data acquisition systems. These have been used for sabotaging the nuclear programs of entire countries.

Common Approach of an Attacker

Information Gathering

The first phase of an attack is information gathering, also known as reconnaissance. An attacker seeks information about the target and tries to find weaknesses in the following areas:

- IT infrastructure (e.g., a known software vulnerability)
- Physical infrastructure and the related access control mechanisms (e.g., breaking into an office, which might have a poor alarm system and allows access to sensitive information)
- Employees, who may have a poor awareness of security
- Processes inside an organization which might be exploitable

Information gathering can be either passive or active. Passive information gathering can be done by searching the web using specialized search queries, such as Google, which can reveal a surprisingly large amount of information. This practice has become known as "Google hacking" or "Google dorking" [WIKI01]. Additionally, social media platforms are an important source of information about employees, especially concerning their telephone numbers, email addresses and other personal information which can be used for social engineering. Using the details of a person allows attackers to personalize their attacks, such as with spear phishing, or sending personalized mails with malicious attachments. These may use, for example, a person's correct name, address, or birthday to create emails with content that sound very familiar or intimate to a victim, thereby increasing the probability that the victim opens the malicious attachment or clicks on a link and visits a malicious website.

Active information gathering includes the use of tools and techniques which interact with the target but increases the risk of becoming recognized as an attacker. Information gathering can be performed in different ways:

- Trying to contact personnel either via email or telephone (vishing)
- Searching a victim's garbage for useful information, such as addresses and telephone numbers [SENG22]. This is known as "dumpster diving".
- Port scans
- Operating system fingerprinting
- (DNS) enumeration (see below)
- Vulnerability scanning
- Collecting useful information that is publicly available (e.g., OSINT)

Techniques such as DNS enumeration might remain undetected, whereas other techniques such as port scanning or vulnerability scanning can often be easily identified by analyzing the log files of servers or firewalls. Network intrusion detection systems (NIDS), which are usually implemented as a security measure in the network infrastructure, analyze incoming traffic for suspicious patterns and raise a security alert if they recognize possible malicious traffic. In particular vulnerability scanners have an easily detectable network fingerprint. Although using scanners can reveal useful information to the attacker very quickly, their use also increases the risk of being detected.

The duration of the information gathering phase can vary considerably and will depend on the motivation of the attacker. A script kiddie can become bored after a few minutes or hours because they just want to play around with a tool they found on the internet, whereas politically motivated attackers might gather information over months before they actually launch an attack. Active information gathering methods will always leave traces but sometimes will only be detected through forensic investigations after an attack has been performed successfully.

Apart from motivation, the type of attack also influences the information gathering phase. An attacker who wants to perform a denial-of-service attack which only affects availability might not need to intrude into a private network and therefore can ignore certain information. However, more information always increases the chances of a successful attack.

Identifying a vulnerable system can be done by enumerating all available services and their versions and afterwards doing a lookup on publicly available exploit databases.

It should be noted that legal public services can automate active constant scanning on the entire internet, updating their databases constantly. Services like these allow searching for unsecured devices which use default passwords and vulnerable services. The services can be used by organizations as well as private persons, making it easier for anyone to find vulnerable systems. They allow searching for specific IP addresses, domains, and web server versions, making this another useful tool for information gathering.

However, information gathering can also be done offline by performing dumpster diving, observation, claiming to be a customer, or infiltrating an organization as an employee. This greatly increases the chance of being detected and is therefore avoided in most cases. Only if an attacker has a very high motivation or where learning about a target may have already failed might it be considered as an approach for information gathering.

Finally, information gathering is a recurring task performed during an attack, because once an attacker has gained access to infrastructure which is not publicly available, they need to keep learning about it.

Exploitation/Gaining Access

Once attackers have gained a reasonable knowledge about their target, they will transition to the actual attack. “Reasonable knowledge” in this context means that they have actually found at least one attack possibility which will have a high probability of success. A successful attack can be caused by:

- Known software vulnerabilities in unpatched software
- Misconfiguration (e.g., missing or wrong configuration)
- Zero-day exploits
- Weak passwords
- Social engineering

If an attack succeeds, the attacker will usually not have an administrator account. This might represent an obstacle to them as they would want to alter the system for their purposes (e.g., by stopping or modifying anti-virus software). Therefore, the acquisition of higher privilege levels is often required after the initial access. Privilege escalation can succeed for the same reasons as the initial attack, such as a software vulnerability or misconfiguration. Misconfiguration is a serious threat to privilege escalation. On UNIX systems, for example, many programs allow access to a root shell and, if they permit the use of SUDO, (an acronym for super user do), they may use this to execute programs as a super user or another user [GTFO22].

While these are attacks that can be executed remotely, it is also possible for an attacker to gain direct access from an organization's internal network. An attacker could be an employee who wants to harm the organization. In addition, an attacker could gain physical access to an office because of insufficient access control and from there might be able to connect to the internal network.

Social engineering is a threat which can lead to the attacker obtaining direct access. It is defined according to NIST as “the act of deceiving an individual into revealing sensitive information, obtaining unauthorized access, or committing fraud by associating with the individual to gain confidence and trust.” [NIST05].

Persisting/Maintaining Access

Gaining unauthorized access to a system often uses exploits that may be difficult to apply, have a high probability of failure or can only be applied at certain times. Therefore, attackers need to maintain access to the compromised system until they have achieved their goals. Again, this depends on the attackers motivation as some may only be interested in a successful attack, but do not want to go further.

Persistent access is usually achieved via rootkit, which is specifically created for this purpose. Rootkits try to maintain access, even if a system is rebooted. Since anti-malware software tries to detect malicious software, rootkits are constructed to hide themselves and additional malware such as key loggers and network sniffers on the system. They can also enable the automated remote control of compromised systems, allowing attackers to create botnets for performing distributed denial-of-service attacks against other systems, or misusing them as a spam server.

Clearing Tracks

Since attacking can have serious legal consequences, attackers want to stay anonymous or at least do not want to be identified in person. As a result, they have a high motivation to remove all traces of their preceding activities after they have achieved their goal. This includes removing all programs and files the attacker has copied to the compromised system(s), clearing, or removing log files, command histories and perhaps destroying the hardware they used for the attack.

Although these are tasks which an attacker performs on completion of an attack, attackers will also use techniques such as proxy chaining, a VPN, or already compromised jump servers during their remote access to obscure their traces.

An attacker must be aware that each activity they perform against a system can be potentially logged, and in most cases, they will be unable to delete all their traces completely.

In the context of security and penetration testing, it is necessary to act in a similar way, since the SUT might be an IDS or an emergency response process.

Incident Response and Post Incident Analysis

The previous sections described an attack scenario from an attacker's perspective. On the other hand, organizations invest in security measures to detect and resolve security incidents, which is a task known as incident response. Note that incident response and the previously described attack phases usually take place at the same time. In many cases, an attacker has already been detected in an earlier phase (e.g., persisting phase), and not just after they have already tried to clear their tracks.

Preparation

While preparation is part of incident management, it is not part of incident response but builds the foundation for a working incident response procedure that will take place in case of a security incident.

Incident response aims to achieve the following:

- Identify an incident and analyze the situation
- Containment, e.g., isolate compromised systems and shut down services
- Eradication e.g., remove compromised user accounts, remove malware, and patch a system
- Recovery e.g., bring services online again and restore data

After an incident has been resolved, there should be a review to identify weaknesses in the infrastructure and in the incident response processes.

Detection and Analysis

The detection of an incident can be either intended or unintended by the attacker. In the case of a website defacement for example, it is the attackers' intention that the attack will be recognized. In other cases, they might leave a message to a system administrator.

Different tools can be used to help in detecting suspicious activities. These include network/host intrusion detections systems (NIDS/HIDS), malware scanners and log analyzers. In the best case, an attack can be identified before the attacker has succeeded. This can be the case if an attacker is too conspicuous during scanning, or if their first attempts at an exploit fail. Also, many failed login attempts, or login attempts from users that do not exist or usually do not log in remotely can be an indication of a potential attack. If this can be detected outside of the organization's network, there is a good chance of defeating the attack.

If suspicious activities are detected inside the organization's network, it must be analyzed which systems are already compromised and how the attacker gained access. This can be done by the IT department, but more commonly a forensics team starts to analyze the incident. The response to an incident is time-critical and actions must take place as early as possible to avoid further harm.

Containment, Eradication and Recovery

If there is a clear picture about which systems are compromised, the next step is to contain these systems to prevent an attacker from compromising further systems. This can be achieved, for example, by shutting down services temporarily, by moving them to another network or locking user accounts.

An important aspect to consider during this phase is that some actions might delete forensic evidence, which could be useful for post-incident analysis. For example, shutting down a system will delete the main memory, which might contain useful data such as the initial exploit that was used. Again, it is vital to react quickly, as an attacker might compromise further systems. The actions performed have to be decided based on the risk level.

Containment and analysis of the situation will alternate at a certain point, as it must be re-evaluated if all systems have been identified correctly, and the attacker is not able to gain further access. If there is sufficient certainty that all affected systems have been identified and contained, eradication can take place. An important aspect during this phase is providing evidence for further analysis. Eradication can include deleting a whole system and recreating it later from a backup. It is also possible to remove only partial components of the system and replace them during the recovery phase with a new component. In both cases, it must be assured that the used backup or component does not contain traces from the previously resolved incident.

Post-Incident Activity

After resolving an incident, several steps must be taken to evaluate and improve the current security routines. This includes:

- Forensic investigations, which in the best case can identify the attacker
- Closing vulnerabilities that might have been revealed through the attack
- Re-evaluating the current infrastructure
- Increasing security awareness of employees
- Re-evaluating and perhaps adapting security policies
- Refining incident response processes
- Making announcements to clients or customers, especially if the organization has reporting obligations. This includes reporting to the corresponding institution or the government.

6. Adjusting Security Testing to Software Development Lifecycle Models - 165 minutes (K4)

Keywords

software development lifecycle

Security Keywords

None

Learning Objectives for Chapter 6:

6.1 The Effects of Different Software Development Lifecycle Models on Security Testing

STE-6.1.1 (K2) Summarize why security testing activities should cover the software development lifecycle

STE-6.1.2 (K4) Analyze how security testing activities are impacted by different software development lifecycle models

6.2 Security Testing During Maintenance

STE-6.2.1 (K3) Define and perform security regression tests and confirmation tests based on a change to a system

STE-6.2.2 (K2) Analyze security test results to determine the nature of a vulnerability and its potential technical impact

6.1. The Effects of Different Software Development Lifecycle Models on Security Testing

The application or system lifecycle can be described as a model with different SDLCs . The most used SDLC phases are planning, analysis, design, development, test, implementation, maintenance and termination.

The activities and planned tasks for each of these phases may differ according to the application, system, project or organization. These are defined in the SDLC model which may be implemented using a sequential development or agile development approach.

Using a sequential development approach, it is easier to recognize the different SDLC phases. With the Agile approach it may not be so clear when and which activity or task from which lifecycle process is performed. The activities and tasks may be frequently repeated, adding value to the application or system with each single iteration.

In the ISTQB Certified Tester Foundation Level [ISTQB FL] syllabus, several development models are mentioned, including the waterfall model and agile software development models (e.g., Rational Unified Process, Scrum, Kanban and Spiral). It is mentioned that security testing should be adapted to these models to be most effective. As may be expected, approaches to security testing also need to be adapted to different SDLC models. This syllabus discusses DevOps in addition to the above-described models.

The STE should have knowledge about the most important characteristics of these SDLC models and how they may impact their ability to perform security testing.

When comparing these categories, differences can be observed regarding the following attributes. A change in any of these attributes will also have an impact on how security testing is performed.

Attribute	Description
Development duration	The time needed from a requirement formulation until deployment • The duration will also affect the allowed time to execute security testing during the SDLC • The less time available the more challenging the choices need to be taken, and priorities set
Deployment size	Larger batches of functionality or a single feature per deployment • Often in direct relation with the development duration • The smaller the size of a deployment, the more specific one can (and must) focus on security testing • With larger deployments, the attack surface may be substantially increased in one iteration. The need for regression testing increases in incremental development models.
Allowed testing time	The amount of time resources reserved to perform testing • In most cases, functional testing may be planned but non-functional testing (including security testing) is often not planned in project lifecycles • If time allows, it can create opportunities for more extensive non-functional testing including security testing
Team independency	The level of security decisions which can be taken by the team • It may allow the team to assign or hire autonomous security testing competence if needed

Attribute	Description
Team cross-functionality	The availability of skills needed (for the development) in the team • The team may have different and complementary security testing competence available
Automation level	How much of the development process is automated. • Much of the security testing may be performed with test automation using SAST- and DAST-related activities
Management principles	Is the organization a line organization, a project, or product oriented? The principles of how the team is managed may influence how security testing is organized. This may be performed with enabling teams, continuous focus on security, just during the project phase, or only during maintenance.
Test environment	A separate and dedicated test environment can be established for destructive security testing.

6.1.1. Sequential Development Models

The complete SDLC is often specified by all processes needed to develop, maintain and support the system from its early initiation until its retirement / disposal. A much-used standard describing all these processes, and their relationships is [ISO 15288].

The system development model describes the implementation of (parts of) the SDLC needed to develop and implement a system. This implementation does not necessarily include all SDLC processes and should be adjusted to the organization (see chapter 5) and the project context.

Information security and security verification should be an integrated aspect covered in all SDLC processes used in the organization. Only then can a truly holistic approach be achieved. This will also support or guarantee that the required activities during the development processes can be conducted in a consistent way.

In [NIST 800-160] the security challenge is describing system security as a design problem. It notes that "a combination of hardware, software, communications, physical, personnel and administrative-procedural safeguards is required for comprehensive security. Software safeguards alone are not sufficient."

This NIST standard presents considerations and approaches covering all SDLC processes on how to address security information activities.

The sequential Waterfall development model or its implementation as a V-model is still a much-used model. The V-model refers to generic testing activities for each of its phases with the goal of shift left. These models are more established in organizations with distinct separation between the different teams and development phases. In general, we may expect the STE to have time to plan, prepare and perform security testing when using this software development model. Phases in the model are scheduled in sequence but may overlap with each other.

In these models, the STE should be aware of the following:

- Security requirements and risks are defined early in a project and should be documented in software requirement specifications.
- Security requirements may change over the course of the project as new threats are discovered, but these may not be reflected in updated software requirements. Security testing may therefore appear to be very specific and complete but may not actually be complete or current due to late project risks.
- Security testing can be performed at any time or in any development phase, but it is common for it to be performed late in the project.
- It may be difficult to address the results of security testing and remediations at the end of a sequential development model project as deadlines will mostly be set at an early phase in the project.

6.1.2. Agile Software Development

Agile software development promotes the completion of work to be done by self-organizing and cross-functional teams in short iterations. Enabler teams which deliver specific services to the project may be available to these Agile software development teams to assist with specific domain competencies, such as security testing.

Generic to Agile software development is that increments (of system and software) are delivered in a series of iterations. Each of these iterations may take from days to some weeks. Agile software development models tend to be used mainly in the application/system development phase, although some models, such as Kanban, can also be applied during the operation phase.

The Scrum framework in various implementations is the most used in Agile software development. All analysis, design, coding, and testing is done during each iteration, including security testing.

Product backlogs act, at least partly, as a requirement specification. Both security requirements and other non-functional requirements are expected to be part of the product backlog. Epics are split into several user stories and tasks which are selected by the team(s) to be developed or delivered in one of the sprints.

Developed functionality may be changed or even deleted in future sprints. The “growing” functionality and future changes or even deletions make an unstable test basis to work with. Agile software development can be seen as a mix of development (new functionality) and maintenance (platform and existing functionality) during the project phase. Repeating automated security tests is therefore essential.

Several approaches can be adopted to performing the security testing activities. Examples are:

- Do some security testing and then shift the focus to functional, technical, or platform related test objects in each different sprint

- Do some security testing in most sprints and perform a complete security test in a dedicated sprint
- Perform all security tests in one (late) sprint which resembles the sequential development model

The Agile development team may involve an enabler team or hire resources to perform the security testing as these competencies often do not reside in the team.

As the solution changes with each sprint regression testing must be performed. Security is not tested or patched into an already built application. Rather, it is achieved through security-oriented design (i.e., security by design) and verification throughout the process of construction

[Synopsys] has described how security testing can be applied in Agile software development by applying the following four principles defined in the Agile Manifesto:

- Developers and testers over security specialists
- Securing as you work over securing after you are done
- Implementing features securely over adding security features
- Mitigating risks over fixing defects.

Developers and Testers over Security Specialists

Experienced security specialists are valuable resources. Agile teams rarely have the luxury of having their own dedicated security specialists. This means that most of the time, Agile teams should be responsible for their own security, and they cannot wait for an external security review before the code moves to the next development phase. Security must be integrated into code development and testing. Teams must own security the same way they own user experience, reliability, performance efficiency, and other non-functional requirements.

Securing as You Work over Securing after You Are Done

Applying secure methodologies and practices when creating, releasing, and maintaining functional software is a must. At the same time, security activities should not force developers to stop what they are doing, go to another tool for remediation, and then come back to what they were doing. The alternative is to integrate security feedback and information into the developer’s tools. Security tasks are presented (e.g., on whiteboards) and priorities set to make them visible alongside other tasks.

Implementing Features Securely over Adding Security Features

The focus should be on delivering the software’s business mission, but the integration of security should always be a consideration. This means that architects, developers, testers and other stakeholders must consider security aspects and work together to define and build more secure systems. Secure systems should be designed and built from the beginning.

Mitigating Risks over Fixing Defects

Risks should be considered which are specific to the business, users, data, and software. Risk management considers the right way to deal with a risk. This may be achieved by taking a high-level view of what could go wrong instead of distilling security down to a long list of individual defects that need to be resolved. Although threat modeling is more difficult than simply localizing and fixing defects, it is an effective approach to detecting problems early in the SDLC. It is definitely cheaper when problems can be resolved before releasing software.

6.1.3. The DevOps Methodology

Most Agile software development covers the delivery of the system or software to the operation department. DevOps goes further by including development and operations. The main objective with DevOps is to deliver (small) changes quickly. In addition, team culture has a much larger impact on the success of the team. DevOps teams are generally more autonomous and more product oriented than other teams.

DevSecOps spans the entire SDLC, including development, security, and operations. During development, security focuses on identifying and preventing vulnerabilities, while in operations, monitoring and defending against attacks are the main objectives.

In general DevOps iterations can be as short as an hour and deliveries are typically single development feature/tasks or small branches. The DevOps team aims to make test results available almost immediately after making a change at each step in the development process. This puts high pressure on the testing and methodologies in use, which in turn has a major effect on the possibilities to perform security testing.

To achieve short DevOps development iterations many of the repetitious and resource-intensive tasks are automated. This is done in the form of a pipeline consisting of a number of pipeline phases, each of which can hold one or more jobs related to performing specific tasks in the build and deployment process. One pipeline phase may be called system testing and may have a job which executes automated regression tests. This pipeline is then run for each feature to be delivered.

DevOps comes in different flavors. The two most used approaches are:

- Using a main or master branch. Changes are developed and tested in a short-lasting feature branch or trunk and directly deployed into production after approval. This is also known as trunk-based development.
- Using a separate development branch allows changes to be delivered into a test environment continuously and deployed together in a small batch after a short period. This is also known as feature-based development.

Security testing activities take time. A common question to be answered involves deciding whether to perform security tests for each pipeline or whether to schedule them during the night after running some pipelines.

DevSecOps is a concept which indicates the importance given to security testing in DevOps. It is often considered as several security testing jobs executed automatically in the pipeline and includes both static analysis (i.e., shift left) and a focus on security-related monitoring and prevention, such as security training and writing secure coding.

Emphasis is placed on security as a team responsibility with security being considered part of all development activities during all phases for everyone in the team.

Common challenges in implementing security testing using the DevOps methodology are:

- Security testing is still considered as a specialized task to be done by specific resources. This inhibits the much-needed integration of security testing within the DevOps team.
- Security may become over prioritized, resulting in other quality characteristics, such as performance efficiency and usability, becoming neglected. Security is important but needs to be implemented in a balanced approach.
- Security may lead to the inhibition of developer creativity, team autonomy and the possibility to experiment. These attributes are considered to be essential for a successful DevOps implementation.

The Phoenix Project [TechTarget] describes the following practices needed for a successful implementation of DevOps. These should also be applied to security testing:

- Create and maintain a flow of tasks
  Security testing is often considered as a large or single task (e.g., application test, network scan, and architecture review). When applying DevOps, it is necessary to plan, prepare and perform security testing in smaller tasks. These should progress (flow) in the same way as other development tasks by applying concepts such as making tasks visible, limiting work in progress, assigning individual tasks to individual people, and automating where possible.
- Ensure instant feedback
  Test results should be available as quickly as possible. They must be understandable and resolvable. The ability to do this is supported by the abovementioned flow and by using smaller tasks. This also helps to reduce technical debt.
- Encourage a DevOps security culture
  The team must be open and transparent regarding security issues. They must be motivated to report security-related issues and consider security as a team responsibility.

Conceptionally, DevOps allows for failing, learning and improving. When introducing DevOps, it is considered better to get started with some small improvements and get the flow running.

To be efficient, the Security Test Engineer should integrate all necessary security-related tasks in the DevOps (CI/CD) pipeline described above. This means not only focusing on the security testing, but also formulating and improving epics, user stories and tasks during the planning phase.

6.2. Security Testing During Operations and Maintenance

6.2.1. Security Regression Testing and Confirmation Testing

Security testing continues after a system is put into production. Changes may occur to the technical environment, to external systems and to SUT integrations. The changes may be due to regular security updates or other changes in middleware, firmware and hardware. In addition, the SUT will be subject to planned and unplanned changes which might open up new vulnerabilities and possible attacks.

All these changes require at least periodic security regression testing. Depending on the size of the changes, a new security test may be needed.

Security tests might involve checking that the system continues to successfully resist attempts to defeat established security controls. Enhancements to usability or performance efficiency are especially prone to negatively impacting security controls.

Security regression testing should focus on confirming satisfying all security requirements and testing for new vulnerabilities that might have been introduced during maintenance activities.

Regression testing is often applied with a collection of test cases that are based on testing individual functions. However, for security testing, it is often insufficient to detect regression defects with a security impact. End-to-end regression testing scenarios are more robust and provide a higher level of confidence that complete transactions can be performed in a secure way. For this type of regression test, a set of security test conditions should be defined and tested each time a change is made to the system. Regression defects can appear from changes in all system relevant parameters. Some of the regression defects may have a security impact.

After a system has been placed into production, additional development effort may be required to correct defects in the released version (i.e., corrective maintenance), to adjust to other changes in the operating environment (i.e., adaptive maintenance), or to extend or enhance features (i.e., perfective maintenance).

The security test perspective for system maintenance focuses on testing changes made to correct defects and core functionality. The purpose of this is:

- to ensure that no new vulnerabilities have been introduced in the SUT
- to verify that existing security defenses are still effective following a change

Part of the maintenance process is to keep firewalls and other security technology current. Continuous system monitoring can detect suspicious activity that may need to be addressed immediately.

7. Security Testing as Part of an Information Security Management System - 105 minutes (K3)

Keywords

None

Security Keywords

Information security management system (ISMS)

Learning Objectives for Chapter 7:

7.1 Acceptance Criteria for Security Testing

STE-7.1.1 (K2) Understand acceptance criteria of security testing and how they influence selecting security testing approaches and test techniques

7.2 Input for an Information Security Management System

STE-7.2.1 (K2) Understand the role of security testing for an effective information security management system

7.3 Improving an Information Security Management System by Adjusting Security Testing

STE-7.3.1 (K3) Evaluate information security management system maturity by bringing in different test approaches, new test objects or improved coverage

STE-7.3.2 (K2) Understand measurability within an information security management system

7.1. Acceptance Criteria for Security Testing

Security testing can be applied as a one-off ad-hoc activity for a system before going into production or as a continuous, systematic process in development. Both types will generate test results, but their ability to provide evidence of significant security risks varies heavily. The same applies to different security test techniques. For example, white-box security testing will generate test results and might identify other vulnerabilities than those generated by using black-box security testing.

Just as with software engineering in general, requirements for security tests are rarely clearly defined, complete, accurate or consistent. Commencing security testing based on such poorly defined requirements might generate some test results, but their value depends on the quality of the requirements which is not predefined. Any actions based on such requirements is likely to be risky for the following reasons:

- Questionable test techniques:
  Security testing uses a specific or a combination of different test techniques, each of which has

strengths and weaknesses. The best test technique does not exist per se, but some preferences exist for achieving a given test objective. Without test objectives all test techniques might match expectations, but without any guarantee to create any added value.

- Questionable coverage:
  Most test techniques do not have a predefined definition of complete but need well defined metrics that have to be met so that the test can be considered done. The type of metrics and their thresholds depend on the test objectives of the security test. Without them, the STE may achieve a level of coverage that might not meet the test objectives.

To avoid these pitfalls, it is essential to define acceptance criteria in advance of any security testing. These must be fulfilled before using the test results as a basis for identifying any deviations or action items.

The word accepted is key. ([WaCh90]) states that this means “that interim and final software products are examined to determine whether they meet specific criteria. If they do, then they have passed acceptance”. Naturally, security requirements should have their own acceptance criteria (cf. ([WaCh90]), which support the acceptance decision to reject, partially accept, or accept.

Security testing can be well suited to control the security acceptance criteria defined for an SUT. The test results, which are usually aggregated in a test report, should contain all necessary information for enabling an acceptance decision. To support this decision-making, the selected security test approach should be based on the specific acceptance criteria. Usually this is done in the following steps:

- Read the security acceptance criteria carefully
- List possible security test techniques (see chapter 2) that can be used to support the acceptance decision. Keep in mind that some security test techniques might cover zero to many acceptance criteria
- Create a specific suite of security tests for these specific acceptance criteria. The guiding principles for this are:
  - Possibility of application:
    Is a specific test technique possible for the SUT? (e.g., are there corresponding tools available?)
  - Optimization of cost, time and quality:
    The challenge is to define a specific test suite and tools that generates significant test results, that can be applied within a given timeframe, and which is cost affective.

After selecting the best security test techniques and tools, the security test must be applied, and the test results analyzed and reported in the test report. The test report itself should reflect the acceptance criteria and form the basis for the security acceptance decision.

7.2. Input for an Information Security Management System (ISMS)

Testing itself does not improve quality. Testing provides information about the quality achieved for a specific quality characteristic, such as security. A test report does not improve the security level. If test report findings are analyzed and most of them are resolved, a confirmation test might be appropriate to demonstrate an increase in security. In addition to these system-specific risk mitigation actions, some of the findings might motivate a specific security policy to avoid such vulnerabilities in the future. On the other hand, some of these findings might have their roots in a lack of security awareness or using immature/non-systematic techniques.

To leverage security testing effectively and efficiently, it must be integrated into an overall security process. It must try to minimize risk and ensure business continuity by proactively limiting the impact of a security breach. This is precisely the goal of an ISMS. [ITGov23b] defines an ISMS as follows:

- An ISMS takes a systematic approach to securing the confidentiality, integrity and availability of corporate information assets.
- An ISO 27001 ISMS consists of policies, procedures and other controls involving people, processes and technology.
- An ISMS is an efficient way to keep information assets secure, based on regular risk assessments and technology- and vendor-neutral approaches.

An ISMS takes a holistic view of security and ensures the effective interaction of the three key attributes of information security:

- Process
- Technology
- Behavior within the organization [Cald11]

The security testing of applications or systems is directly related to technology. However, each vulnerability identified by security testing might have its roots in process and/or behavior and might be mitigated in the future by changing processes and/or behavior.

There are at least the following reasons for an organization to implement an ISMS which are directly supported by security testing [Cald11]:

- Strategic, to better manage information security within the context of overall business risks
- Customer confidence, to demonstrate that an organization complies with information security management best practices
- Regulatory, to meet various regulatory requirements
- Internal effectiveness, to tactically manage information more effectively within an organization.

Security testing plays an important role in establishing an ISMS. Since it relates to testing, it demonstrates the status quo of a system. This can be understood as follows:

- As evidence for a goal, which is planned to be reached
- As evidence of a starting point, which motivates further security actions

A well-known feedback loop for modeling the goal and starting point is the Plan-Do-Check-Act cycle (PDCA). ISO 27001 "adopts the PDCA process model, which is applied to structure all ISMS processes" [Cald11]. Both aspects, goal and starting point, are visible within this model:

- Goal:
  After the Plan and Do steps, the Check step establishes whether the planned goal has been reached.
- Starting point:
  The result of the Check step, which is supported by security testing activities, is analyzed within the Act step to improve the overall process. Its deviations are the basis for the next PDCA cycle.

Security testing provides the most added value within an organization if it considers both aspects (i.e., goal and starting point)

To measure achievement of the Plan and Do steps in terms of improved security, a security test approach must be precisely adjusted, so that the security test exactly matches the planned goal. It is good practice to define acceptance criteria for security tests for the Check step in advance when defining the security plan.

To leverage security testing as the starting point for the next PDCA cycle, the test technique, the applied tools, the executed test suites and all test results, (positive ones as well as negative ones), must be reported within the test report.

It may be possible to leverage the same security test approach for setting a new starting point, and for measuring the effectiveness of the next PDCA cycle. Security testing must be applied in a very systematic approach. All relevant parameters must be stored to allow for a repeatable, objective security testing.

7.3. Improving an ISMS by Adjusting Security Testing

The effectiveness of a defined ISMS in providing increased security is highly dependent on the proactive actions introduced by an ISMS. This means that an ISMS should derive security controls based on the current security status, and an assessment or a current business situation (e.g., an incident).

In addition to any direct risk mitigation action that is done to correct issues identified within normal operations, the ISMS tries to derive controls that prevent such issues from occurring in future development. The more iterations a specific ISMS has experienced PDCA iterations, the better the set of deviated security controls and the better the security level of all developed applications.

7.3.1. Improving the Holistic View of an ISMS

To improve an ISMS by increasing its holistic view, security testing must take on a new responsibility in addition to setting the baseline for Check step within the PDCA cycle. If the goal is to improve the ISMS maturity, the STE must bring in completely new aspects that were not yet planned as part of the PDCA cycle. These new security tests might generate additional insights into the SUT, which can then be used to further improve ISMS maturity by deriving additional security controls.

Typical dimensions that an STE can use for enhancing ISMS scope are:

- Additional test objects:
  Each system that must reach a specific level of security has to be considered within its typical environment when it is in production. The system may be located behind a firewall, or connected to a central database, or have an API interface to an external applications/system, or controlled by a daily backup process, or have a connection to a privileged accounts management system. The more systems that are connected to the SUT, the broader the attack surface. Each system can be attacked, and if it is broken there is a high probability that the overall network of systems fails as well.

  An ISMS should cover as many aspects as possible to manage information security as holistically as possible: Each aspect should reflect possible attack vectors. To do this, it is essential for an STE to focus on as many test objects as possible, because any one of them might be a risky component in an overall network. If one of the security tests identifies additional weaknesses in a component that is not yet part of the overall ISMS, it can improve the maturity of the ISMS based on this new information. It is not the STE's task to define countermeasures (e.g., security policies or process adjustments), but their task does include being open-minded regarding additional SUT components being useful for maturing the ISMS.

- Additional test approaches:
  Another possibility for an STE to bring in additional insights into a system is to use different types of testing. Even if the Check action as part of the PDCA cycle focuses on a dynamic black-box test of the SUT, it might be beneficial to perform a static test as well. This could potentially show additional vulnerabilities that have not been identified so far and that could also be used by attackers. These new insights should be leveraged as input to an ISMS to further improve its maturity.

- Improved coverage:
  Even when remaining with the existing test object and test approach, the STE can generate valuable insights for maturing the ISMS. Simply by adding some more test cases could generate completely new insights. This can easily be done by using structured fuzz test tools or rainbow tables.

  Another way to improve the coverage might be to enhance the number of test cases executed per unit of time (e.g., by automating some test suites) or increase the number of test cycles performed to enforce unusual behavior that can be used for attacks. If these improved coverage measures identify additional vulnerabilities, it will help improve the ISMS maturity.

7.3.2. Improving Measurability within an ISMS

The STE can improve ISMS maturity by introducing a metrics-based feedback loop. These metrics are usually named key performance indicators and support continuous improvement using PDCA cycles. The fundamental idea of the underlying PDCA cycle is to check the effectiveness of preceding Plan and Do actions. The more objective this Check Act can be done, the more objective the feedback loop becomes. The policies deriving from ISMS include coverage metrics for test techniques and behavior. If a policy from an organization's overall policy portfolio includes the direct use of a test technique, then security testing can directly check whether that policy is successful.

Even if a policy only indirectly touches the processes used, the STE can still support the measurement of its effectiveness. For example, it might be difficult to directly measure the effectiveness of some training given regarding secure coding conventions. One way could be to conduct examinations at the end of each training session. Another way would be to set up a security test that precisely checks for the occurrences of security anti-patterns which had been covered in the training. The more frequently these anti-patterns still exist after the training, the less value the training added in terms of security.

The same could be said when evaluating the effectiveness of unwanted behavior. For example, the security test might define a phishing simulation which counts the number of unwanted clicks made within an email. When considering phishing, higher click rates are generally seen as being bad because it means users fail to notice the email is phishing, while low click rates are often seen as good. However, to measure the effectiveness of an awareness initiative, this security test must be repeated to enable a before and after change to be measured. [StGrTh20]

Security testing can improve ISMS maturity because feedback loops are based on the hard facts generated by security testing. Improvements take place more quickly and are more reliable than those based on subjective or feelings.

8. Reporting Security Test Results - 135 minutes (K3)

Keywords

ethical hacker, risk mitigation, test report

Security Keywords

None

Learning Objectives for Chapter 8:

8.1 Security Test Reporting

STE-8.1.1 (K2) Understand the criticality of security test results and how this affects their handling and communication

8.2 Identifying and Analyzing Vulnerabilities

STE-8.2.1 (K3) Evaluate test results from a given security test to identify vulnerabilities

8.3 Close Vulnerabilities

STE-8.3.1 (K3) Evaluate different techniques for closing identified vulnerabilities

8.1. Security Test Reporting

Each security test ends in a test report [ISTQB Glossary]. Without test reporting, test lacks evidence that can be used to determine actions or decisions based on the test result.

The following standard information is important when reporting a failed security test case:

- Used test environment: This usually includes specific IP-addresses, applied IP whitelisting, and used accounts/passwords.
- Preconditions of the executed tests. This includes all preparation activities to be applied before the prepared test suite can be executed. This might include activities such as log-in-details, specific configuration file settings or specific perimeter configurations.
- Used test data
- Procedure of test execution
- Expected results and actual results

A failed security test means a specific test has detected the violation of at least one security aspect from the CIA triad (see Chapter 1). A good test report includes a sufficient level of detail to enable the test to be repeated. Tools used to execute the tests might be named in the test report and screenshots might be included to support the test results with evidence.

In general, security test reports should be handled with high level of confidentiality. If this type of information is leaked outside the organization, it could dramatically reduce the organization's reputation. Even worse, the information could be used to attack any systems which include this vulnerability.

The more failed tests a security test report contains, the more critical and sensitive the test report and its communication is. In general, every security test report must be communicated with care within the organization. This includes internal communications within the organization producing the SUT, as attackers might come from inside an organization (cf. [SwissCybInst20]). On the other hand, security test reports might be important for many people within an organization. This paradox directly influences the STE's reporting activities and is usually resolved by creating different versions of the same test report, each of which contains different levels of detail. Each version of the test report should follow the "need to know" concept. This applies to those whose task it is to mitigate identified risks and therefore receive a complete test report or fragments of it based on the "need-to-know" concept.

The sensitivity of a security test report may be modified according to vulnerabilities identified. This is essential when ethical hackers identify a vulnerability in an SUT and wish to inform only the developer to give them the opportunity to mitigate this risk before the test report is made publicly available. This responsible disclosure is one of the characteristics of an ethical hacker, especially for white-hat hackers [Huneidy21]. Grey-hat hackers often use test report publication to increase pressure on an organization to work on patches [Huneidy21].

8.2. Identifying and Analyzing Vulnerabilities

It is important to note that the absence of a failed test suite does not mean that the system is without defects. Even passed test suites do not necessarily mean that the examined attack vector cannot be exploited. It simply states that with the test suites used it is not possible to be exploited by an analyzed attack vector.

If a security test fails, a potential vulnerability is identified. The test report should give all evidence needed to repeat the failed test case. A security test report might demonstrate many vulnerabilities. The following steps must be taken before any remedial action is taken:

- Vulnerability demarcation:
  Usually a failed test represents a single failed test case and represents one vulnerability. However, several other test cases might exist which show the same vulnerability. For example, if an empty input parameter can be used to take over control of an application, maybe the same behavior can be achieved when using a 100MB file as an input parameter. During the vulnerability demarcation phase, different but similar tests are executed to demarcate the identified vulnerability. This is important for the subsequent risk assessment and must be supported by the STE.

- Adjusting risk likelihood:
  This step is performed to double-check the risk likelihood of being able to identify a vulnerability in production. Usually, a security test is not performed on a production system. Even if the test environment is similar, it will never be completely identical to the production environment. In particular, some security controls might be explicitly disabled to allow a security test to be

performed. If an SUT demonstrates a vulnerability, it might be obfuscated by other parameters put in place for production. If this is the case, the vulnerability still exists but cannot be directly exploited due to other parameters. This adjustment might change the risk level suggested by the security test and might therefore change the overall necessity to plan risk mitigation actions. The STE is tasked with considering this risk adjustment and taking corresponding actions.

- Adjusting risk impact:
  This step is performed to double-check the possible risk impact arising from the exposure to a vulnerability. Usually, the STE's focus is on technical aspects, which makes the calculation of business impact difficult to estimate. Especially if the STE reuses impact assessments for identified vulnerabilities from an outside source (see chapter 4, CVSS), they can be imprecise for a specific context. If a vulnerability is identified, the business stakeholders refine the possible risk impact. The vulnerability may be estimated to have no impact from a business perspective (e.g., if the impacted component is seldom used), or it might be considered to have a high level of business criticality. This adjustment might change the risk level suggested by the STE and warrant an update to planned risk mitigation actions.

If all three of the above steps are applied, a clear view can be obtained of the identified vulnerability and its identified risk. The adjustment of risk likelihood and risk impact must consider some important parameters:

- Close communication with business stakeholders, as they have the final say when discussing possible risk impact
- Close communication with the operations team, as they have the final say when considering production parameters and how they differ from those used in security testing
- Even if the existence of the vulnerability in the SUT is clear, stakeholders may try to actively influence the risk adjustment step (by adjusting likelihood or impact)

If the remaining risk level is considered to be too high to go into production or stay in production, a risk mitigation plan should be created. Management is responsible for deciding about the urgency of such risk mitigation plans. The decision can be between the following levels of urgency:

- Stop operation immediately or stop further go live activities:
  If the risk level is considered to be too high and cannot be accepted, it can only be avoided by not running the SUT. The level of risk (e.g., high, medium or low) that influences this decision depends on the risk appetite, which is defined as "*the amount of risk that an organization is willing to accept to achieve its objectives*" [CARM22]
- Continue running the system with intensive monitoring:
  If the system is too critical or the risk is too critical, the SUT might continue running but is intensively monitored
- Add risk mitigation actions to the normal release plan:
  If the risk can be handled or the system has many strict release constraints, the risk mitigation actions are analyzed and performed but not directly applied to the SUT. Instead, the patched

components are added to the normal release cycle to ensure that the next planned release contains the required security patches.

The STE must ensure that confirmation tests and regression tests are performed for each risk mitigation action. The confirmation test should consider the evidence provided in the test report and should also use the lessons learned from the vulnerability demarcation step.

8.3. Close Identified Vulnerabilities

If an identified vulnerability is known in terms of vulnerability demarcation, and if it is decided to mitigate the risk, at least two high-level alternatives are available for mitigating the identified vulnerabilities:

- Hide the vulnerability by reducing expected risk
- Avoid the vulnerability by patching the affected system

Both alternatives can be combined. Hiding the vulnerability may provide a short-term solution and the proper patch can be done afterwards.

8.3.1. Hiding a Vulnerability

The idea of hiding vulnerabilities is similar to the approach a tester applies, when they differentiate between a defect that does not cause a failure, and a defect that causes a failure, (i.e., that can possibly impact a customer).

Even if the risk adjustment steps have shown that the identified vulnerability is caused by a failure, (i.e., it exposes a high risk), the system itself might be changed in such a way that the defect does not reveal itself to the customer, even if the defect remains unchanged. In terms of risk mitigation, this type of action targets the reduction of risk impact. The defect still exists in the system, but it cannot be exploited anymore. Typical approaches for hiding vulnerabilities in this way are:

- Traffic blocking:
  Modern firewalls allow for very sophisticated analyses and blocking mechanisms. If the traffic needed to exploit the vulnerability is well understood, many firewalls can be configured to block such traffic patterns. By doing so the vulnerability remains unchanged, but it can no longer be exploited.
- Virtual patching:
  Virtual patching does not necessarily block traffic, but it converts it in a way that the vulnerability cannot be exploited. [OWASP11] defines this as “a security policy enforcement layer which prevents the exploitation of a known vulnerability. The virtual patch works since the security enforcement layer analyzes transactions and intercepts attacks in transit, so malicious traffic never reaches the application. The resulting impact of a virtual patch is that, while the actual source code of the application itself has not been modified, the exploitation attempt does not succeed.”

- Switching off or reconfigure specific system features
  If the vulnerability has a very limited scope within the system, it might be possible to switch off the functionality affected by the identified vulnerability.
- Reduce scope of vulnerabilities
  It might be possible to accept the risk associated with a vulnerability by reducing its scope. For example, it might be possible to set up IP filters so that only well-known machines with dedicated IP addresses can connect to the vulnerable machine. In addition, it may be possible to disable the vulnerable machine from external access and to allow only internal access.

The STE is responsible, when confirmation testing, for applying all hidden vulnerability approaches to ensure that the specific vulnerability cannot be exploited anymore.

8.3.2. Avoiding a Vulnerability

Generally, avoiding the vulnerability is a very time consuming and expensive action. The following steps must be performed to avoid a vulnerability:

Step	Description
Locate the vulnerability	• As the risk mitigation action must be taken at the level used to implement the functionality (e.g., code, models, and configurations) it might take time to identify the affected component and within that component the affected area based on an identified vulnerability at the system level
Understand the vulnerability	• Before making a repair, a full understanding of the vulnerability must be obtained by analysis of the vulnerability in the affected area (e.g., code snippet)
Identify the risk mitigation action	• An approach for risk mitigating must be developed. • The mitigation action might consist of a completely new algorithm, a new component, a small configuration change or only some minor code adjustments (e.g., including a specific exception behavior).
Execute the risk mitigation action	• The identified risk mitigation action is applied.
Confirmation test	• Perform a confirmation test on the system to test if the vulnerability has been eliminated.

Step	Description
Regression test	• Testing is a fundamental part of avoiding vulnerabilities and should not only focus on the changed code. It is essential that the complete regression test suite is executed to make sure that the system is still running correctly, and the mitigation action has had no unwanted side effects.
Deploy	• Deploy the patched system. • After the system is deployed there is usually some close monitoring for a period to be sure that the system is running well.

9. Security Testing Tools - 90 minutes (K3)

Keywords

dynamic application, interactive application security testing, static application security testing, vulnerability scanning

Security Keywords

open-source software, software composition analysis (SCA)

Learning Objectives for Chapter 9:

9.1 Categorization of Security Testing Tools

STE-9.1.1 (K3) Analyze different use cases and apply categorizations for security testing tools

9.2 Selecting Security Testing Tools

STE-9.2.1 (K2) Understand the usage and concepts of dynamic security testing tools

STE-9.2.2 (K2) Understand the usage and concepts of static security testing tools

9.1. Categorization of Security Testing Tools

There are several possibilities to categorize security testing tools. A selection of these categories includes:

- The activity in the security test process where the tool can be used
- Open-source versus closed source security testing tools
- Static analysis versus dynamic test tools
- Platform / infrastructure versus the application
- Security test execution versus security test management
- Black-box testing versus white-box testing versus grey-box testing

Each of these categories has its advantages and disadvantages when the STE is selecting the most appropriate security testing tool. This syllabus presents three principal categories:

- Black-box testing versus white-box testing
- Static security testing versus dynamic security testing
- Open-source versus closed source security testing tools

It is expected that the STE will build their own library of tools for their domain and contexts. Nevertheless, they may still adopt the suggested categories.

9.1.1. White-box Security Testing Tools

If the STE has code level access, white-box security test tools provide them with knowledge relating to the code, configuration information, libraries used, applications, system and platform, architecture, and login details. One important prerequisite is for the STE to have permission to perform the analysis, since they will be trying to identify and assess the vulnerabilities of the system and integrated systems.

9.1.2. Black-box Security Testing Tools

A prerequisite for performing black-box testing is access to a running application or system in a production-like environment. This is necessary to be able to execute the tests using black-box security test tools, which consider the SUT as a black box and do not need any internal knowledge of the software. Black-box security testing tools place their focus on identifying vulnerabilities when running the application / system.

9.1.3. Grey-box Security Testing Tools

Performing grey-box security testing gives the STE some limited information about the application / system internals and access to a running version. Grey-box security test tools can be seen as a mixture of white-box security testing tools and black-box security testing tools. They need some internal information as well as a running system. The focus is to identify vulnerabilities by running the application / system and executing tests which consider internal details.

9.1.4. Static Security Testing Tools

An important dimension for security test tool categorization is based on the difference between static security testing and dynamic security testing. Definitions, differences, and descriptions of static and dynamic security tests are discussed in section 2.1.2.

Within this category, the tools can be considered according to their use. These include network testing, operating system testing, database testing and application testing.

Static security testing has much in common with white-box security testing. The main difference is that static security testing tools do not need the application or system to be running. The tool accesses the code, libraries or configuration files in scope and analyzes them with regard to the tool's internal repository of known syntax, semantics or code standard vulnerabilities for the particular language in use.

There are various ways to perform security tests with tools. For static security testing tools these include, but are not limited to, SAST and SCA. These are explained in more detail below.

Static Application Security Testing
SAST is a typical static security testing activity mostly included in a Dev(Sec)Ops pipeline as discussed in chapter 6.1 and [NIST DevSecOps]. Within these pipelines it runs automatically whenever some code has changed and is checked in. Using SAST that way gives immediate feedback. SAST is primarily focused on the application and in most cases does not cover any platform or infrastructure components. In addition, these tools give good code coverage metrics.

There is a strong dependency between static security testing and the tool attribute open-source software, as open-source tools by definition deliver the source code. That means that SAST can be performed for all open-source systems. This is not the case for closed source applications. It might be possible to have some static testing performed (e.g. identifying some reused libraries), however this type of analysis is not possible due to obfuscation or by special license agreements prohibiting static analyses.

Software Composition Analysis
Software composition analysis (SCA) has many touch points with security. SCA tools analyze vulnerabilities in the code, including the dependencies and the open-source components used by the application. These components are well known and may have several vulnerabilities. SCA tools can suggest security vulnerability remediations based on the identified components. When doing this, almost all SCA tools use the Common Vulnerabilities and Exposures [CVE21] database of publicly disclosed vulnerabilities. (see section 4.2.2)

9.1.5. Dynamic Security Testing Tools

Dynamic security testing tools interact with the SUT while it is running. Both black-box security testing and grey-box security testing are closely linked with dynamic security testing. The tools may be considered in the categorizations of DAST and IAST.

Dynamic Application Security Testing
As with SAST, dynamic application security testing (DAST) is commonly used in a DevSecOps context [NIST DevSecOps]. The testing activity is performed automatically in the pipeline using a configurable black-box security testing tool. This analyzes the application or simulates an attacker while the software is running, looking for vulnerabilities such as input validation, fuzz testing, authentication and authorization, configuration and deployment, session management, error handling, and cryptography.

The DAST scan simulates different real-time attacks on the SUT in an automated fashion to identify any vulnerabilities in the application. Test techniques are used on a running SUT when performing dynamic testing. As a result, these are more time and performance consuming compared to static testing. Even though DAST focuses on the application, the identified vulnerabilities often relate to the infrastructure components which are necessary to run the application.

DAST does not reliably cover all OWASP Top 10 [OWASP Top 10] or SANS CWE Top 25 issues [CWE21]. Many tools can cover specific aspects of each of the vulnerability classes, but a false sense of security can emerge from using these tools.

Interactive Application Security Testing

Interactive application security testing (IAST) is a hybrid test approach which leverages both static and dynamic security testing. The tools are used to determine if known vulnerabilities in the source code are exploitable during runtime.

An agent is installed in the environment where the application is running which monitors the application and identifies any vulnerabilities in the application while the STE (or dynamic security testing tool) is interacting with the SUT.

9.1.6. Considerations for Selecting Security Testing Tools

The STE must know which tool to select for which context. They should therefore have knowledge of the different categorization schemes and tools belonging to each category.

Security testing tool catalogues may help to select the right tool. Some examples can be found in: [KALI], [OWASP], [SANS] and [NIST]. Note, however, that tools may be categorized differently in some of these catalogues and may be present in one and missing in another.

The STE does not need to know about and be able to operate all security testing tools available. STEs who have been active for a longer time in one particular domain / context would typically build their own specific libraries.

When selecting a tool or building a tool library, it is recommended to do the following:

- Focus on what needs to be verified
- Do not become dependent on a single supplier for your required test results. Use multiple suppliers for the same tool functionality.
- Periodically scan the market for new emerging tools.

Open-Source vs Closed Source

The difference between open-source and closed source (licensed) test tools can be an important aspect of tool selection.

Anyone can participate in the development of open-source applications or tools. This helps to eliminate security vulnerabilities as quickly as possible, at least if the open-source project has an active development community. In addition, open-source tools can be proofed, (at least theoretically), so that backdoors would be visible in the code and can be excluded.

Open-source software can be customized and used for specific contexts, giving it a clear advantage.

The following characteristics might be considered as disadvantages of using open-source tools:

- Missing professional support especially when no active community supports a specific product. However, some organizations specialize in providing support for OSS applications, which is an important aspect for the enterprise environment.
- Licensing issues (e.g., when using GNU's Not Unix public licenses)
- It is necessity to have necessary development skills available (e.g., to adjust to specific contexts)
- If there is a vulnerability in an open-source system, there is a risk that someone will identify it and will create exploits
- Further development of OSS applications is uncertain, as they are mostly community driven.

In contrast to OSS applications, closed source applications use proprietary code that is not available to the user. This offers the advantage of being offered support contracts by the supplier.

The following characteristics might be considered as disadvantages of using closed source software:

- License fees have to be paid

- There is no guarantee against backdoors
- Any security vulnerabilities could remain unknown for a long time
- Strong vendor reliance may occur where a customer becomes heavily dependent on a specific vendor's products or services, making it difficult to switch to a different vendor
- In general, there is only limited ability to customize the tool to a specific context (e.g., the code cannot be modified).

9.2. Applying Security Testing Tools

9.2.1. Understand the Usage and Concepts of Static Security Testing Tools

The following aspects describe some of the principal characteristics of static security test tools:

- Are more effective if knowledge of the SUT exists
- Can be applied even if the application itself can still be incomplete and contain defects
- Are strongly related to white-box testing
- Well suited for finding unsecured code or misconfigurations early in the SDLC, since they only require static information of the source code
- Covers the complete source code and configurations and therefore requires read access to them
- Able to read the given object, such as the source code of an uncompiled application, and compare this with predefined data sets of best practices and known vulnerabilities (e.g., unsecured commands and chains within the source code)
- Automation makes them cost effective
- Often provide false-positive results since they are not context aware. This means they do not know the use cases, call stack and composition of multiple lines of code. As a result, they might identify vulnerabilities in code that will never be executed in real-life and therefore do not have any negative impact on security.

Using, for example, network scanning, SAST tool would assess the configurations files to find unsecured configurations.

9.2.2. Understand the Usage and Concepts of Dynamic Security Testing Tools

Continuous application security is important in the context of the continuous integration and delivery approach found in Agile software development. Security needs to be continuously and repeatedly verified for each increment. The motivation for using dynamic test tools is to have them running on a regular basis and applying automation to ensure that the latest known vulnerabilities are immediately found and reported. Continuous security monitoring and improvement is also called DevSevOps, (see chapter 6 and [NIST DevSecOps]).

DAST typically focus on OWASP’s top 10 vulnerabilities such as: injections (e.g., SQL injection, cross-site scripting, and command injection), broken access controls, cross-site-request-forgery, race conditions, business logic errors, memory leaks, and known vulnerabilities.

It is also important to mention that automated DAST tools only scan a set of predefined attack vectors. Therefore, other testing and security checks are still mandatory.

Using the example of network scanning, a dynamic test tool first scans the network for open ports, then runs services under those ports.

10. References

- [Bittau] Cryptographic protection of TCP Streams (tcpcrypt) https://tools.ietf.org/html/draft-bittau-tcp-crypt-04
- [BSI21] https://www.bsi.bund.de/EN/Themen/KRITIS-und-regulierte-Unternehmen/Weitere_regulierte_Unternehmen/LuSi/Luftsicherheit_node.html, accessed on November 7, 2022
- [Bullock2017] Jessey Bullock: "Wireshark for Security Professionals: Using Wireshark and the Metasploit", Wiley 2017, online available via https://computerscience.unicam.it/marcantoni/reti/laboratorio_wireshark/Wireshark%20for%20Security%20Professionals%20-%20Using%20Wireshark%20and%20the%20Metasploit%20Framework.pdf
- [BURP22] https://portswigger.net/burp, last accessed on 1. December 2022
- [Cald11] Alan Calder, Jan van Bon: "Implementing Information Security based on ISO 27001/ISO 27002 – a management guide", Van Haren Publishing, 2011
- [CAPEC21] CAPEC: "Common Attack Pattern Enumeration and Classification: A Community Resource for Identifying and Understanding Attacks", online available via https://capec.mitre.org/
- [CARM22] Mary Carmichael, CRISC, CISA, CPA, Member of ISACA Emerging Trends Working Group: "Risk Appetite vs. Risk Tolerance: What is the Difference?", online available at https://www.isaca.org/resources/news-and-trends/isaca-now-blog/2022/risk-appetite-vs-risk-tolerance-what-is-the-difference
- [CERT]: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487865
- [CERT1]: https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices
- [CIS22] https://www.cisecurity.org/cis-benchmarks/ , last accessed on 1. December 2022
- [CLAIR22] https://github.com/quay/clair , last accessed on 1. December 2022
- [Cloudflare] https://www.cloudflare.com/learning/security/glossary/what-is-zero-trust/ , last accessed on 1. December 2023
- [CVE21] CVE® Program Mission: "Identify, define, and catalog publicly disclosed cybersecurity vulnerabilities.", online available via cve.org
- [CVSS21] First, Improving Security together: "Common Vulnerability Scoring System SIG", online available via https://www.first.org/cvss/
- [CWE21] CWE: "Common Weakness Enumeration: A Community-Developed List of Software & Hardware Weakness Types", online available via https://cwe.mitre.org
- [CWSS21] CWE: "Scoring CWEs: Common Weakness Scoring System CWSS", online available via https://cwe.mitre.org/cwss/cwss_v1.0.1.html
- [FUZZDB22] https://github.com/fuzzdb-project/fuzzdb , last accessed on 1. December 2022

- [Gart21] Gartner Information Technology: “Glossary“, online available via https://www.gartner.com/en/information-technology/glossary
- [GITLAB22] https://docs.gitlab.com/ee/user/application_security/sast/ last accessed on 1. December 2022
- [GTFO22] https://gtfobins.github.io/, accessed on November 13, 2022
- [Huneidy21] Mariangel Huneidy: “The Ultimate Guide to Ethical Hacking “, Sept 21, online available via The Ultimate Guide to Ethical Hacking (0x1.gitlab.io)
- [IETF23] Internet Engineering Task Force: “Introduction to the IETF”, online available via https://www.ietf.org/about/introduction/
- [ISO 15288] ISO/IEC 15288 – System Life Cycle Processes.
- [ISO 25010] ISO/IEC 25010:2011
- [ISO 27001] ISO International Standards Organization: “ISO/IEC 27001:2013; Information technology — Security techniques — information security management system— Requirements”
- [ISO 31000], ISO 31000:2018 - Risk management
- [ISO/IEC/IEEE 29119-3], Software and systems engineering — Software testing – Part 3 – Test Documentation
- [ISO/IEC/IEEE 29119-4], Software and systems engineering — Software testing – Part 4 – Test Techniques
- [ISO_Web_21] ISO International Standards Organization: “Consumers and standards: Partnership for a better world”, online available via https://www.iso.org/sites/ConsumersStandards/1_standards.html
- [ISTQB FL]: ISTQB Certified Tester Foundation Level Syllabus v4.0 Certified Tester Foundation Level (CTFL) v4.0 [NEW!] (istqb.org)
- [ISTQB Glossary]: International Software Test Qualification Board: Glossary https://glossary.istqb.org/en/search/
- [ISTQB_ATTA_SYL]: ISTQB Certified Tester Advanced Level Technical Test Analyst (CTAL-TTA) Syllabus Technical Test Analyst (istqb.org)
- [ISTQB_ATA_SYL]: ISTQB Certified Tester Advanced Level Test Analyst (CTAL-TA) Syllabus Test Analyst (istqb.org)
- [ITGOV23a] IT-Governance: “ISO 27000 Series of Standards”, available via https://www.itgovernance.co.uk/iso27000-family
- [ITGov23b]: “ISO/IEC 27001 – Information Security Management: The international standard for information security”, in IT-Governance, available via https://www.itgovernance.co.uk/iso27001
- [KALI] https://www.kali.org/tools/ , last accessed 2. December 2022
- [KUBEAUDIT22] https://github.com/Shopify/kubeaudit, last accessed on 1. December 2022

- [METASPLOIT22] https://www.metasploit.com/ , last accessed on 1. December 2022
- [Micro09] Microsoft: "The STRIDE Threat Model", 12/2009, via The STRIDE Threat Model | Microsoft Docs
- [Micro22] Microsoft: Zero Trust Essentials eBook, via Zero Trust Essentials eBook (microsoft.com)
- [MITRE21] The MITRE Corporation: "Media FAQs", 2018, via media-FAQs-2018.pdf (mitre.org)
- [NESSUS22] https://tenable.com/products/nessus , last accessed on 1. December 2022
- [NIST DevSecOps] https://csrc.nist.gov/Projects/devsecops
- [NIST Glossary] https://csrc.nist.gov/glossary
- [NIST 800-160] https://csrc.nist.gov/pubs/sp/800/160/v2/r1/final
- [NIST SP 800-161] Supply Chain Risk Management Practices for Federal Information Systems and Organizations
- [NIST02] https://csrc.nist.gov/glossary/term/security_service, accessed on November 6, 2022
- [NIST03] NIST Special Publication 800-61, Revision 2, Computer Security Incident Handling Guide
- [NIST05] https://csrc.nist.gov/glossary/term/social_engineering, accessed on November 13, 2022
- [NIST06] NIST SP 800-137, Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations, accessed on September 2011
- [NISTIR 7007]: An Overview of Issues in Test Intrusion Detection Systems
- [NMAP22] https://https://nmap.org/ , last accessed on 1. December 2022
- [OPENVAS22] https://www.openvas.org/ , last accessed on 1. December 2022
- [OSI] https://opensource.org/definition/
- [OWASP byte code obfuscation]: https://owasp.org/www-community/controls/Bytecode_obfuscation
- [OWASP Test Guide]: https://owasp.org/www-project-web-security-test-guide
- [OWASP Top 10] https://www.owasp.org
- [OWASP] https://owasp.org/www-project-web-security-test-guide/v41/6-Appendix/A-Test_Tools_Resource, last accessed 2. December 2022
- [OWASP11] Ryan Barnett, Dan Cornell, Achim Hoffmann Martin Knobloch "Virtual patching Best Practices", 2011, shared by OWASP, online available at https://owasp.org/www-community/Virtual_Patching_Best_Practices
- [OWASP21] OWASP: "The Open Web Application Security Project", via https://owasp.org/
- [OWASP24] OWASP Dependency-Check. https://owasp.org/www-project-dependency-check/

- [PCI DSS Chapter 6]: https://www.pcidssguide.com/pci-dss-requirement-6
- [PCI22] https://listings.pcisecuritystandards.org/documents/PCI-DSS-v4_0.pdf, accessed on November 7, 2022
- [RFC4765] The Intrusion Detection Message Exchange Format (IDMEF), Network Working Group
- [SANS] https://www.sans.org/tools/, last accessed 2. December 2022
- [SecJour21] Security Journey: "Why vulnerability list methodologies matter". Available via https://www.securityjourney.com/post/why-vulnerability-list-methodologies-matter-and-why-we-trust-cwe-owasp
- [SENG22] https://www.social-engineer.org/framework/information-gathering/dumpster-diving/, accessed on November 10, 2022
- [Shacklett] Shacklett, M. E., *What is* an attack vector*?* https://www.techtarget.com/searchsecurity/definition/attack-vector
- [SNYK22] https://snyk.io/ , last accessed on 1. December 2022
- [SONAR22] https://www.sonarqube.org/, last accessed on 1. December 2022
- [SQLMAP22] https://sqlmap.org , last accessed on 1. December 2022
- [SSLSCAN22] https://github.com/rbsec/sslscan , last accessed on 1. December 2022
- [SSLYZE22] https://github.com/nabla-c0d3/sslyze , last accessed on 1. December 2022
- [Stallings18] Stallings William, Brown Lawrie, 2018, Computer Security: Principles and Practice, ISBN 9781292220611
- [StGrTh20] Michelle P. Steves, Kristen K. Greene and Mary F. Theofanos: "*Categorizing Human* Phishing Detection Difficulty: A Phish Scale", Journal of Cybersecurity. Published online Sept. 14, 2020. Available via https://academic.oup.com/cybersecurity/article/6/1/tyaa009/5905453
- [SwissCybInst20] Swiss Cyber Institute: "41 Insider Threat Statistics You Should Care About", Sept 21, online available via 41 Insider Threat Statistics You Should Care About - Swiss Cyber Institute
- [SYNOPSIS] https://go.synopsys.com/software-integrity-agile-security-manifesto.html
- [TechTarget] TechTarget: "The three ways: The Phoenix project", available via https://www.techtarget.com/whatis/definition/The-Three-Ways
- [TELETRUST] https://www.stand-der-technik-security.de/startseite/
- [TERRASCAN22] https://runterrascan.io/, last accessed on 1. December 2022
- [TR02021] https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html
- [UNECE20] https://unece.org/sustainable-development/press/un-regulations-cybersecurity-and-software-updates-pave-way-mass-roll, accessed on November 7, 2022

- [HIPAA] https://www.cdc.gov/phlp/publications/topic/hipaa.html, last accessed on 22 August 2023
- [WaCh90] Dolores R. Wallace, John C. Cherniavsky: "*Guide to Software Acceptance*", NIST Special Publication 500-180, 1990, online available via https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-180.pdf
- [WAFEC]: https://owasp.org/www-project-wafec/
- [WIKI01] https://en.wikipedia.org/wiki/Google_hacking, accessed on November 10, 2022
- [WIKI02] https://en.wikipedia.org/wiki/Stuxnet, accessed on November 10, 2022
- [WIRED21] https://www.wired.com/story/solarwinds-hack-supply-chain-threats-improvements/, accessed on November 6, 2022
- [ZAP22] https://www.zaproxy.org/ , last accessed on 1. December 2022

Appendix A – Learning Objectives/Cognitive Level of Knowledge

The following learning objectives are defined as applying to this syllabus. Each topic in the syllabus will be examined according to the learning objective for it.

The learning objectives begin with an action verb corresponding to its cognitive level of knowledge as listed below.

Level 1: Remember (K1)

The candidate will remember, recognize and recall a term or concept.

Action verbs: Recall, recognize.

Examples
Recall the concepts of the test pyramid.
Recognize the typical objectives of test.

Level 2: Understand (K2)

The candidate can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify and give examples for the test concept.

Action verbs: Classify, compare, differentiate, distinguish, explain, give examples, interpret, summarize

Examples	Notes
Classify test tools according to their purpose and the test activities they support.	
Compare the different test levels.	Can be used to look for similarities, differences or both.
Differentiate test from debugging.	Looks for differences between concepts.
Distinguish between project and product risks.	Allows two (or more) concepts to be separately classified.
Explain the impact of context on the test process.	
Give examples of why test is necessary.	

Examples	Notes
Infer the root cause of defects from a given profile of failures.	
Summarize the activities of the work product review process.	

Level 3: Apply (K3)

The candidate can carry out a procedure when confronted with a familiar task or select the correct procedure and apply it to a given context.

Action verbs: Apply, implement, prepare, use

Examples	Notes
Apply boundary value analysis to derive test cases from given requirements.	Should refer to a procedure / technique / process etc.
Implement metric collection methods to support technical and management requirements.	
Prepare installability tests for mobile apps.	
Use traceability to monitor test progress for completeness and consistency with the test objectives, test strategy, and test plan.	Could be used in a LO that wants the candidate to be able to use a technique or procedure. Similar to 'apply'.

Level 4: Analyze (K4)

The candidate can separate information related to a procedure or technique into its constituent parts for better understanding and can distinguish between facts and inferences. A typical application is to analyze a document, software or project situation and propose appropriate actions to solve a problem or task.

Action verbs: Analyze, deconstruct, outline, prioritize, select.

Examples	Notes
Analyze a given project situation to determine which black-box or experience-based test techniques should be applied to achieve specific goals.	Examinable only in combination with a measurable goal of the analysis.

Examples	Notes
	Should be of form 'Analyze xxxx to xxxx' (or similar).
Prioritize test cases in a given test suite for execution based on the related product risks.	
Select the appropriate test levels and test types to verify a given set of requirements.	Needed where the selection requires analysis.

Reference

(For the cognitive levels of learning objectives)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and

Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon B – Business Outcomes traceability matrix with Learning Objectives

The table below shows the syllabus Business Outcomes and number of Learning Objectives.

Business Outcomes: Security Test Engineer		Number of LOs
ID	Description	
STE-BO1	Understand the fundamental security paradigms, and their impact on security testing	6
STE-BO2	Use and apply appropriate Security Test techniques and know their strengths and limitations	7
STE-BO3	Contribute to planning, designing, and executing Security Test	5
STE-BO4	Understand how Security Test standards and security best practices can be utilized for Security Test	4
STE-BO5	Adjust and perform Security Test activities accordingly to specific organization context	3
STE-BO6	Adjust and perform Security Test activities accordingly to specific development methods and software development lifecycles	4
STE-BO7	Feed Security Test results into an information security management system (ISMS) for an active security risk management	4
STE-BO8	Collect, evaluate, and aggregate test results, write a detailed test report with all evidence and findings	3
STE-BO9	Based on a needed Security Test approach identify proper requirements for tooling and assist in the selection of Security Test tools	3

The following table shows the traceability between Business Outcomes and Learning Objectives:

			Business Outcomes								
LO Number	Learning Objective	K-Level	STE-BO1	STE-BO2	STE-BO3	STE-BO4	STE-BO5	STE-BO6	STE-BO7	STE-BO8	STE-BO9
STE-1.1.1	Explain different security levels of assets and their corresponding protection level	K2	x								
STE-1.1.2	Explain the relationship between information sensitivity and security testing	K2	x								
STE-1.2.1	Describe the role of security testing in the context of security audits	K2	x								
STE-1.3.1	Explain the concept of zero trust	K2	x								
STE-1.3.2	Apply zero trust in security testing	K3	x								
STE-1.4.1	Exemplify the concept of open-source software reuse in software development and its impacts on security testing	K2	x								
STE-2.1.1	Give examples for security test types according to a black-box, white-box and grey-box security context	K2		x							
STE-2.1.2	Give examples for security test types according to static security testing or dynamic security testing	K2		x							
STE-2.2.1	Apply security test cases, based on a given security test approach, along with identified functional and structural security risks	K3		x							
STE-2.2.2	Describe how to test reconciliation and recertification for identities and permissions	K2		x							
STE-2.2.3	Describe how to test identity and access management control	K2		x							
STE-2.2.4	Describe how to test data protection control	K2		x							
STE-2.2.5	Describe how to test protective technology	K2		x							

STE-3.1.1	Explain different activities, tasks, and responsibilities within a security test process	K2			x						
STE-3.1.2	Understand the key elements and characteristics of an effective security test environment	K2			x						
STE-3.2.1	Give examples for security tests on the component test level based on a given code base	K2			x						
STE-3.2.2	Give examples for security tests on the component integration level based on a given design specification	K2			x						
STE-3.2.3	Implement an end-to-end security test which validates one or more security requirements related to one or more business processes	K3			x						
STE-4.1.1	Explain different sources of test standards and best practices and their applicability	K3				x					
STE-4.2.1	Apply the concept of the Open Web Application Security Project, Common Vulnerability Enumeration, Common Weakness Enumeration, the Common Vulnerability Scoring System and the Common Weakness Scoring System and how to leverage them for security testing	K3				x					
STE-4.3.1	Explain Pros and Cons of test oracles used for security testing	K2				x					
STE-4.3.2	Understand Pros and Cons of using security best standards and best practices	K3				x					
STE-5.1.1	Analyze a given organizational context and determine which specific aspects to consider for security testing.	K3					x				
STE-5.2.1	Analyze the impact of governmental regulations on security policies.	K3					x				
STE-5.3.1	Analyze an attack scenario (attack performed and discovered) and identify possible sources and motivation of the attack.	K4					x				

STE-6.1.1	Summarize why security testing activities should cover the software development lifecycle	K2						x			
STE-6.1.2	Analyze how security testing activities are impacted by different system development lifecycle models	K4						x			
STE-6.2.1	Define and perform security regression tests and confirmation tests based on a change to a system	K3						x			
STE-6.2.2	Analyze security testing results to determine the nature of a security vulnerability and ist potential technical impact	K2						x			
STE-7.1.1	Understand acceptance criteria of security testing and how they influence selecting security testing approaches and techniques	K2							x		
STE-7.2.1	Understand the role of security testing for an effective information security management system	K2							x		
STE-7.3.1	Evaluate ISMS maturity by bringing in different test approaches, new test objects or improved coverage	K3							x		
STE-7.3.2	Understand measurability within an ISMS	K2									
STE-8.1.1	Understand the criticality of security testing results and how this affects their handling and communication	K2								x	
STE-8.2.1	Evaluate the results from a given security test to identify security vulnerabilities	K3								x	
STE-8.3.1	Evaluate different techniques for closing identified vulnerabilities	K3								x	
STE-9.1.1	Analyze different use cases and apply categorizations for security testing tools	K3									x
STE-9.2.1	Understand the usage and concepts of dynamic security testing tools	K2									x
STE-9.2.2	Understand the usage and concepts of static security testing tools	K2									x

Appendix C – Release Notes

ISTQB® Security Test Engineer is a new release. For this reason, there are no detailed release notes per chapter and section.

Appendix D – Security Testing Domain-Specific Terms

Term Name	Definition
authentication	A procedure determining whether a person or a process is, in fact, who or what it is declared to be.
authorization	Permission given to a user or process to access resources.
encryption	The process of encoding information so that only authorized parties can retrieve the original information, usually by means of a specific decryption key or process.
firewall	A component or set of components that controls incoming and outgoing network traffic based on predetermined security rules.
information security management system (ISMS)	A part of an overall management system, based on a business risk approach, to establish, implement, operate, monitor, review, maintain and improve information security.
information sensitivity	A measure of the importance of protecting information assigned by its owner (After NIST).
open-source software	Software that can be accessed, used, modified, and shared by anyone.
Open Web Application Security Project (OWASP)	OWASP, the Open Web Application Security Project, lists common weaknesses and is pretty famous for publishing their OWASP Top 10 ranking.
privilege escalation	The exploitation of a bug or flaw that allows for a higher privilege level than what would normally be permitted (After NIST)
rootkit	A set of tools used by an attacker to gain and maintain root-level access to a host to conceal the attacker's activities through covert means. (After NIST)
security policy	A high-level document describing the principles, approach and major objectives of the organization regarding security.
security service	A capability that supports one or many security goals. (After NIST)
social engineering	An attempt to trick someone into revealing information (e.g., a password) that can be used to attack systems or networks.
software composition analysis (SCA)	A practice for identifying open-source and closed source components in use in an application, their known security vulnerabilities, and adversarial license restrictions (After NIST)

Term Name	Definition
STRIDE	An acronym for six threat categories (i.e., spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege) used to model potential threats to a system.
zero-trust	A model designed to minimize uncertainty in enforcing accurate, least privileged per request access decisions in a component, system and services if a network is viewed as compromised. (NIST Glossary)

Index

All terms are defined in the ISTQB® Glossary (http://glossary.istqb.org/).