A4Q
Security Essentials

Syllabus 2019

Version 1.0: 03.08.2017
Final Version 2019 1.5: 23.08.2019

Table of Contents

0. Introduction

0.1 Purpose and business outcomes

The Microsoft Security Development Lifecycle (MSDL) [URL:MSDL] approach as well as the Open SAMM [URL:OpenSAMM] assessment model suggest that any staff member within a software project should receive comprehensive basic training on IT security and advanced training according to the role held within the organization. The goal of this syllabus is to provide a comprehensive introduction to IT security for any team member involved in the development of an IT system, software application or embedded system. Staff trainings following this syllabus will fulfill basic security training needs as recommended in [ISO/IEC 27034-1] and the MSDL. It addresses the needs of project managers, requirement engineers, software developers and testers alike.

General benefits of the training are:

- understand the most common security related terms, concepts and processes.
- actively take part in and contribute to security related risk management activities

Specific benefits for project managers:

- align project activities with required or recommended security related activities
- understand and explain fundamental security requirements that a given system must meet
- understand and explain the activities required for developing a specific system or application in a secure manner

Specific benefits for developers:

- understand and explain the activities that are required for developing secure systems and applications
- understand common security related mistakes in development

Specific benefits for requirement engineers:

- understand and explain how fundamental security requirements can be established
- understand common security related mistakes in requirements engineering

Specific benefits for testers:

- understand the role of testing as part of a security development lifecycle
- understand and explain different security testing types

Specific benefits for IT risk managers and IT security experts[1]

- receive guidance on what to include in a comprehensive basic training on IT security

[1] This is actually an intended benefit of the syllabus per se. Members of this group may have participated in advanced trainings on security and hence may not require basic training.

1. Fundamentals of IT Security (110 minutes)

Basic Terms

accountability, asset, authenticity, authentication, authorization, availability, confidentiality, integrity, non-repudiation, personally identifiable information (PII), security, threat, threat agent, vulnerability, weakness

Learning Objectives

(1.1.1) Recall properties of information security and related attributes. (K1)

(1.2.1) Identify examples for assets, vulnerabilities and threats or threat agents. (K2)

(1.2.2) Recall causes for a weakness and when a weakness becomes a vulnerability. (K1)

(1.2.3) Identify which system property from confidentiality, integrity or availability is affected by a given system weakness, vulnerability or action from an attacker. (K2)

(1.2.4) Describe typical activities in a security risk management process. (K2)

(1.2.5) Identify typical human threat agents/attacker types. (K1)

(1.3.1) Identify secure development principles based on authentication, authorization and auditability. (K2)

(1.4.1) Recall which properties may differentiate security relevant system failure from other types of system failure. (K1)

(1.4.2) Given a list of root causes and common errors, identify which are typical for security vulnerabilities. (K2)

(1.5.1) Given a short description of a security standard, name the type of the security standard. (K1)

1.1 The Concept of IT Security (15 minutes)

In this syllabus we discuss security in the context of information systems. By definition an information system is a closed or open dynamical, electronic system, with the ability to store and process information [Eck 14].
According to [ISO 25010] security is *"The degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorisation."*

The 'security-triangle' CIA is named by virtually all leading IT-Security organizations (e.g. OWASP, InfoSec) as one of the most important security patterns.

CIA stands for the quality attributes confidentiality, integrity and availability:

- Confidentiality in an information system means that access to information must be restricted to those authorized ([ISO 25010]). Data must be protected against non-authorized access, no matter if the unauthorized access happens on purpose or by accident.
- Integrity is defined as "degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data" ([ISO 25010]). Note that information is more than data. For example, retrieving an encrypted file may be a breach of integrity, whereas an unauthorized retrieval of the information stored in the file is a breach in confidentiality.
- Availability means [ISO 25010] that the information stored in a system or the services delivered by the system are available to authorized users, when its use is required. In other words, the system must be operational. Referring to [ISO 25010] availability can also be considered as a property of reliability.

Other properties of security according to [ISO 25010] are:

- Non-repudiation – a system must have appropriate mechanisms (e.g. logging) in place, so that it can be proven that actions and events have taken place. These mechanisms (e.g. the logs) must be protected against tampering.
- Accountability – the "degree to which the actions of an entity can be traced uniquely to the entity". This usually requires some form of authentication and storing e.g. identifiers for users, programs, processes and systems in log-files.
- Authenticity – measures the "degree to which the identity of a subject or resource can be proved to be the one claimed".

1.2 Assets, Threats and Vulnerabilities – The Context of IT Security (55 minutes)

Ensuring the security of an IT system involves more aspects than the definition for security given above implies. It encompasses services and information that are valuable to a company. The European Union Agency for Network and Information Security (ENISA) defines security as "All aspects related to defining, achieving, and maintaining data confidentiality, integrity, availability, accountability, authenticity, and reliability. …" [ENISA]
However, considering security always implies that a system might be at risk for some reason.
A system is said to be secure, when it is able to adequately protect specified assets from unauthorized access, destruction, disclosure, modification of data, and/or denial of service. Any circumstance or event that may result in one of the latter outcomes is called a threat to the system [ENISA].

An asset can only be defined in the context of an organization: "Anything that has value to the organization, its business operations and their continuity, including Information resources that support the organization's mission." ([ENISA])
Examples for assets within an information system can be personally identifiable information (PII) for individuals, such as credit card numbers or any information that may be used for authentication. For a company it may, for instance, be electronically stored business plans, information on pending patents, salary lists, any trade secret, or contracts. Company assets also may include computer-controlled production capabilities or other services delivered via the Internet. For a nation, assets may include military secrets, critical infrastructure such as telecommunication, water and electrical power supplies and traffic control systems.

Attackers or adversaries (e.g. individuals, organizations) can threaten assets[1].
Adversaries will usually exploit a weakness in an information system. A weakness can arise from a programming error, bad design or architecture or faulty requirements [ENISA]. Very often misconfigurations result in weaknesses.
A vulnerability in a system arises when an attacker can gain or has access to a weakness.

Typical weaknesses and vulnerabilities may impact the above named security triangle CIA as follows

- Confidentiality, e.g.:
  - The Heartbleed bug in the open SSL encryption library allowed an attacker to read out arbitrary memory contents from web servers including passwords and keys.
  - Misconfiguration of servers or firewalls may leak program version or IP-address information to an attacker that he can use to prepare future attacks. The action of collecting this type of information, is called banner grabbing or fingerprinting.
  - Error messages may reveal program internal information e.g. on its structure, the programming language used or configurations that an

[1] Also chance events, such as a hardware failure or natural catastrophes threaten assets. These are not discussed in this syllabus.

attacker can use to prepare future attacks.

- Integrity, e.g.:
  - Leaving default passwords in place may easily allow an attacker to change the configurations of a router or upload spoofed firmware to an IoT-device.
  - Deleting server logs or wiping hard drives to obscure an attack.
  - An unsuitable input validation may allow an attacker to embed malicious script code in the guestbook of a website[1].
- Availability, e.g.
  - Denial of Service (DOS) or distributed denial of service attacks (DDOS) may overload a server with queries so that its services are no longer available.
  - Flaws in a programs logic or database queries can allow an attacker to craft inputs that will result in the system using up a huge amount of computational resources. The attacker may use this to incapacitate the system. This is also a DOS type of attack.
  - An attacker uses ransomware to encrypt a hard drive. This also impacts the integrity of the system, since the attacker tampers with its storage facilities.

Note: An attacker might use social engineering techniques to obtain access to a vulnerable part of a system (see chapter 2 for details).

Building and maintaining secure systems requires a continuous risk management following e.g. [ISO 31000:2018] that includes at least the following activities:

1. Identify protection goals, e.g. assets that need protection, privacy regulations that must be followed, etc. Identify generic threats to these assets. To do this, an organization must understand its business context and know which information systems are actually used.
2. Define which level of protection is required for each system or component.
3. Analyze the architecture, systems, components, libraries, etc. in use and identify actual threats and vulnerabilities.
4. Act upon the findings.
5. Validate if the implemented actions succeed in eliminating identified threats or mitigate the risks imposed to an acceptable level.

[ISO 31000:2018] stresses the cyclic nature of risk management. Hence, the process activities must be repeated as required.

Risk management processes for security have to be implemented on different levels within an organization. [NIST-SP-800-53] gives an example for an organization wide security risk management process. An example for a threat assessment process that is fit for use in a security software development lifecycle can be found in [Howard 06].

An important factor in a security risk analysis is to understand the potential threat agents, that is who or what poses a threat to the system. [CC-Part-1] gives as examples:

[1] Cross Site Scripting or XSS – see chapter 2.

- Hacker
- Malicious user
- Non-malicious user
- Computer processes
- Accidents

The human threat agents can be categorized by
- Skill,
- Motivation,
- Relative position to the IT-System.

A categorization by skill may differentiate
- Script-kiddies have at most basic programming skills but know how to operate a computer system and use existing programs or scripts (hence the name) that exploit known vulnerabilities.
- Dedicated hackers have good programming skills, knowledge of network and operating system technologies and good background-knowledge of IT-security concepts and tools.
- Skilled hackers have excellent programming skills including knowledge of hardware-near programming languages (e.g. assembler) and are able to write their own security tools, exploits or – in case of malicious intent – malware.

Skilled hackers may also be hired by large organizations such as corporations, a countries secret service or its military. If this is the case, the threat is usually much larger than from a single or even a group of skilled hackers. These organizations have vast resources to back up any type of attack on information systems including human intelligence.

A categorization by motivation may distinguish
- Curiosity – usually a trait of non-malicious users (e.g. software testers)
- Showing-off
- White hat or ethical hacker – finds vulnerabilities, detects ongoing attacks and informs vendors, organizations and the public
- Hacktivists – use cyber-attacks as a mean for protest, defacement of political enemies or to steal and publish secret information
- Cracker, black hat hacker or cyber-criminal – executes or supports illegal attacks on IT-systems with malicious intent (e.g. financial interests, espionage, terrorism or revenge)
- Terrorism, asymmetric warfare and espionage – typically orchestrated by groups of skilled hackers[1]

A categorization by position should cover at least
- Out-side attacker – an entity that may start its attack from outside the local network, not as a legitimate user and without prior inside information
- Inside attacker – a legitimate user within the organization that may have malicious[2] or non-malicious intent (e.g. curiosity).

[1] These types of intent often involve so called Advanced Persistent Threats (APT) – a set of stealthy and continuous computer hacking processes aimed at specific targets, e.g. critical infrastructure

[2] Note: An inside attacker may have been coerced into his activities by blackmail or other forms of social engineering.

- Man-In-The-Middle – controls or sniffs network traffic using an infected device, e.g. a computer, router, server or smart-phone that may be within or outside the local network.

1.3 Principles in Developing Secure Software (15 minutes)

The preceding section is concerned with what a secure system should prevent. This section is about which principles must be built into software to be secure:

- Authentication: The process of establishing an identity (and thus authenticity) is called authentication. A secure system must have trustworthy methods for authentication. These can be for instance knowledge-based (e.g. passwords) token-based, use challenge-responses (e.g. captcha's, tokens sent via SMS or e-mail) or biometric attributes or they can employ trusted third parties or identity providers (e.g. an organization issuing an electronic passport with a chip and a pin). In a multi-factor authentication, more than one of these methods is applied.
- Authorization: A secure system must ensure at all times that all actions by processes and entities are executed within the limits of predefined rights. Typical principles to be applied include [Saltzer 75]
  - Complete mediation: Any access to data objects, processes, services, etc. is checked every time for proper authorization.
  - Least privilege: A program, user, system, etc. can only access exactly those resources required for its legitimate purpose
- Auditability: Auditability of a system is a prerequisite to accountability and non-repudiation. Secure systems must have mechanisms to log events that may be part of an attack and protective measure against tampering with the log or the logging mechanisms. [Lamport 78] states that auditor-functionality should include reliable error-handling procedures. A safe and secure system must be able to audit its own state to a certain extent.

Security practitioners often describe the triplet authentication, authorization and auditability as 'gold-standard', since all three words start with the syllable 'au' – the chemical symbol for aurum (gold). These Au-properties were originally discussed in [Lamport 78]. Any implementation of security functionality must be done in a way that prevents tampering. Therefore, it is highly recommended to use a trusted computing base for security functionality. A trusted computing base should be certified independently after the Common Criteria Standard [CC-Part-1] to [CC-Part-3].

1.4 Why Security is different (15 minutes)

Typical failures are deviations from an expected behavior that can be easily observed by a user or a software tester. Although security functions may also show these kinds of failures, the majority of security relevant defects may not even be noticed during normal use of a system. Seemingly correct functional behavior may have side effects that go unnoticed for the majority of users and even testers. E.g. a function may allow additional operations that are not described in its specification. Such weaknesses in the software implementation or flaws in the system design and architecture may be exploited by an attacker. Also, faulty compilers, compiler options, deprecated third party libraries or hardware may give rise to vulnerabilities. Other vulnerabilities result

from weaknesses e.g. in protocol implementations, flawed memory management by an application or the operating system itself. Uncovering or preventing this kind of defects requires a deep technical understanding.

Typical reasons for security vulnerabilities and related defects are:

- Lack of security awareness, naivety and being unsuspicious
- Lack of due diligence, e.g. in compliance to regulations and in checking and keeping systems and configurations up-to-date
- Missing, incomplete or inadequate security policies or insufficient control whether policies are actually being adhered to
- Missing security requirements and concepts
- Flaws in the system architecture or its design
- Lack of usability in security functionality, e.g. insecure defaults
- Wrong assumptions, e.g. assuming that other security measures are successful, that only certain inputs will be received, or that a communication channel is secure.

Security must therefore be considered in each software development phase, as well as on a regular basis for systems in operation.

1.5 Security Standards (10 minutes)

Standards offer a collection of requirements defined by subject matter experts as well as process and document templates. They also can be used as checklists and so help to avoid common mistakes.
There are hundreds of standards on information security and related topics in existence – much more than can be meaningful applied. To give a comprehensive overview, this syllabus distinguishes four categories:

- Regulations, e.g. laws, directives or regulations such as the European General Data Protection Regulation (GDPR).
- Standards for secure software development as SDLC (Software-Development Lifecycle) models covered in [ISO/IEC 27034] and implemented by the Microsoft Security Development Lifecycle. Another example is the Common Criteria catalogue, (see e.g. [CC-Part-1], [CC-Part-2], [CC-Part-3]) which includes good practices for security requirements as well as procedures for verification and validation of security functions.
- Standards on Information Security Management Systems such as [ISO/IEC 27001] or [NIST-SP-800-39]. These standards describe risk management processes and how they can be applied to people, processes and IT systems to keep information assets secure.
- Domain specific standards. These are either industry specific like the Payment Card Industry standards or technology related, such as the standards and guidelines published by the Open Web Application Security Project (OWASP).

2. Understanding Attacks (255 minutes)

Terms

Access control, attack surface, attack vector, CSRF, DDOS, DOS, Exploit kit, Man-in-the-middle, Phishing, Replay attack, Root kit, Scareware, Smishing, Sniffing, Trojan, Virus, Vishing, Worm, XSS, Zero-day exploit

Learning Objectives

(2.1.1) Recall typical patterns used to classify attacks. (K1)

(2.2.1) Know the characteristics that define virus, worm, trojan, scare-ware, root kit and exploit kit. (K1)

(2.3.1) Explain attack surfaces and attack vectors. (K2)

(2.3.2) Explain the principles of memory-based attacks. (K2)

(2.4.1) Recall the four layers of the Internet protocol suite and their purpose. (K1)

(2.4.2) Describe typical attack surfaces and vectors for a client-server session. (K2)

(2.4.3) Understand that attacks can happen on all layers of the Internet protocol suite. (K2)

(2.4.4) Understand the concept of injection attacks and possible defenses. (K2)

(2.4.5) Differentiate how DOS and DDOS work. (K2)

(2.4.6) Differentiate how different XSS types work. (K2)

(2.4.7) Differentiate how a CSRF works. (K2)

(2.4.8) Differentiate possible causes for vulnerabilities in authentication, session management and access control in web applications. (K2)

(2.4.9) Understand the risks of using weak or no encryption. (K2)

(2.4.10) Differentiate the principles of symmetric and asymmetric encryption and the need to establish authenticity in web communication. (K2)

(2.5.1) Describe the steps in the social engineering attack cycle. (K2)

(2.5.2) Recall characteristics of phishing, vishing and smishing. (K1)

(2.6.1) Describe typical threats to wireless communications (K2)

(2.6.2) Describe principles and examples for operating wireless communications in a secure manner (K2)

2.1 Overview on Attack Taxonomies (5 minutes)

Attacks or – more specific – attack vectors may be classified by

- Type of attacker – see chapter 1.2
- Type of malware that is used – see section 2.2 for examples.
- Attack surface, which is the interface to the exploitable application. See section 2.3 for details.
- Vulnerable entity or property within the software, e.g. a broken authentication management, script execution, etc. A more elaborate description is given in section 2.4.
- Method used in the attack, as in Denial of Service, Injection or Social Engineering – see sections 2.4. and 2.5.

There are also vulnerability classification schemes such as STRIDE, DREAD and 'Common Vulnerabilities and Exposures' in use. These are discussed in chapter 3 of this syllabus.
A special case is the zero-day-exploit. It uses a vulnerability as attack vector that was unknown before the attack occurred – so no patch exists.
Current real world examples for the attack types discussed in this chapter can be found for instance under [URL:CISA alerts] or [URL:NVD].

2.2 Malware Types (15 minutes)

Malicious software – in short malware – describes any type of software that is written, installed and executed with malicious intent.
The following list describes typical forms of various malware types. Actual malware can also be a combination of multiple types:

- A virus is a type of malware that requires a host program for execution. It cannot run without another program but replicate itself – i.e. infect other programs, files, computers, etc.
- A worm is a type of malware that can run on its own, replicate and typically marks and identifies already infected systems or computers. Usually consists of several parts or segments like a real worm. Worms are used to spread malware from machine to machine throughout a network.
- A Trojan is a type of malware that is covertly installed on a machine, often with other, non-malicious software and may be able to hide its operations. Trojans usually have installer capabilities and install or download other malware, such as key loggers, rootkits, viruses, etc.
- Exploit kits can be installed e.g. on servers and are capable to automatically identify and exploit software vulnerabilities in client machines communicating with the server. They upload and execute malicious code on the clients in the process.
- A root kit is designed to gain unauthorized – usually administrative – access on target machines without being noticed by surveillance programs such as a regular system monitors.
- Scareware is a type of malware employing social engineering techniques. It manipulates users into buying unneeded and mostly useless software by causing anxiety or the perception of a threat.

2.3 Attack Surface (30 minutes)

[Howard 06] defines the attack surface as “the sum of all code and functionality accessible to users and potential attackers.”
From a testing perspective [Whit03] differentiates generic types of attack surfaces:

- the User Interface (UI)
- the Application Programming Interface (API)
- the file system
- the system’s memory

Attacks via any attack surface may allow

- Different types of injections
- Controlling directory paths, URLs, etc. for reading and writing
- Unauthorized access to data or services
- Direct calls to operating system functions

Attacks via the user interface can occur via any input that is directly manipulated by a user. This includes attacks against weak authentication mechanisms, e.g. usage of default passwords, weak authorization concepts where confidential information can be accessed in several ways with different rights (see section 2.4 for further discussion and examples), injections like SQL injections (also see section 2.4) or buffer overflows (see memory based attacks below).

Attacks using an API are specific to the API. Often known vulnerabilities of older and unpatched API versions or common API programming errors are exploited in this type of attack. Often side-effects or function overloading of API functions are exploited. This may be difficult to spot in design and implementation, since the code gives the impression, that only admissible functions and constructs have been used. A well-known example is the use of the Perl open-command with two parameters (instead of three) which may allow for shell code injection or DOS.

Typical attacks on the file system include reading information from unencrypted files, tampering with saved data, configuration files or executable files, disassembly of binary files or decryption of weakly encrypted files, e.g. MD5-hashtables of passwords.

Memory based attacks make use of the layout and operations of a computer’s memory. To understand how memory attacks work, a basic understanding of the layout of a computer’s memory is required.
Typical basic memory structures are stack and heap. Both will contain data, instructions and jump or return addresses. Return addresses indicate where instructions are located that should be executed next. Data stored in the computer’s memory will typically include user-controlled input that is submitted via user interface, a file or an API. Data written or queried by a user typically have a predefined maximum length. Two common patterns applied in memory-based attacks use this simple fact:

- Buffer overflow: The adversary writes to the memory an amount of data larger than what the program is expecting. In this way he can change the content of subsequent data fields. He may be able to manipulate return addresses or control variables and thus change the way the program

executes. He also may be able to embed his own malicious code as data entries and bring it to execution.[1]

- Buffer over-read: An adversary can read from memory positions, that he should have no access to. This may be the case, if the adversary requests (e.g. in a query) a larger set of data than the program expects. From a buffer over-read, an attacker may gain unauthorized access to any kind of data, including additional knowledge of source code running on a server, content of control variables, return addresses and their position in memory to prepare future attacks.

Both types of attack are possible, if the principle of complete mediation is not consequently applied on the implementation level.

Note:
An adversary will often use more than one attack surface to stage an attack. e.g. enter a crafted string via the user interface that will corrupt a return address in the computer's memory and then will execute machine readable code, embedded in the string.

2.4 Common attacks and web security (160 minutes)

Attacks on the security of information systems have already been performed a long time before the advent of the Internet. Torturing an adversary's messenger, reading his secret messages, listening to his radio transmissions or wiretapping into a telephone line, are just a few examples. Actually, many of the first Internet hackers were so-called 'phone phreaks' – people that actively searched and exploited weaknesses in telephone networks for various reasons, e.g. making free calls (fraud), showing-off or sometimes just technical curiosity [MIT03].
To speak of common attacks in this chapter may sound a little bit preposterous, since a multitude of different attack patterns is known. MITRE's Common Attack Pattern Enumeration and Classification (CAPEC™) identifies for instance more than 500 [URL:CAPEC]. It is obvious that these cannot be discussed in a first course on IT security. That is why this chapter confines itself to representative attack types that provide typical examples on how antagonists act.
The Internet provides a worldwide interconnectedness that enormously magnifies the reach of potential attackers. To understand IT security requires a basic understanding on how communication in this giant network operates.

2.4.1 The Internet Protocol Suite

Communication within the Internet requires hardware components and software that controls various aspects of communication.
The Internet protocol suite (in short TCP/IP) is a set of specifications ([RFC 1122], [RFC 1123]) that describe how communications are established using a four layer architecture.

The four layers are

[1] Advanced forms call library functions (e.g. return-to-libc-attacks) in malicious way or use snippets of the programs own code (return-oriented programming: ROP). These are not discussed here.

- The link layer establishes communications between a host (e.g. a desktop, a local server, a tablet, etc.) with its direct neighbourhood. This neighbourhood is the directly-connected or 'local network'[1]. There are actually other software layers below the link layer. They establish physical connections – e.g. between network cards or different hardware components in a computer.
- The Internet layer protocols are used to transport data packets (datagrams) between hosts in different local networks. The protocols are 'connectionless', which means that there is no delivery-guarantee for a datagram.
- The transport layer provides end-to-end communication for applications running on hosts. Two major protocols are used on this layer. The first is the state- or connectionless User Datagram Protocol (UDP). The second is the Transmission Control Protocol (TCP). The latter is a reliable protocol that establishes a confirmed connection via a handshake transaction and keeps track of all data packages sent by sequencing.
- The application layer incorporates two protocol categories. The first are protocols that provide services directly to users such as telnet, ftp and smtp. The second are support protocols that support the protocols used by the users. An example for a support protocol is the Domain Name System (DNS). It supports the location and identification of computer services and devices by human-readable names instead of abstract addresses.

2.4.2 Client-Server sessions on the web

A session is an exchange of information between two devices like computers. Web sessions between a client and a server are typically handled by application layer protocols. The setup of the communication as well as the actual exchange of data packages is handled by the lower level protocols.
The encryption protocol TLS (transport layer security) is used to wrap and encrypt application layer data before it is sent using a transport protocol like TCP. To show that a protocol uses TLS the letter 's' for secure is added to the protocol name, as in https or sftp.
A typical web-session is initiated by a client that sends a request to a webserver, e.g. via a browser. The request is processed on the webserver – for example via scripts for database access. Then an answer such as a status of the request or data is returned to the client.

2.4.3 The Internet Protocol Suite as Attack Vector

The original purpose of the Internet protocol suite was to specify reliable, efficient and easy to implement ways of communication. The designers did not have security as a major requirement in mind. For this reason, many original protocol specifications are flawed from a security point of view. Many protocols have no secure mechanism for authentication and authorization. Also, secure protocol specifications may contain design flaws, or their implementations can contain vulnerabilities. Mending these issue-types in the specifications is an ongoing process.

Vulnerabilities can occur on any of the four layers. For instance, on the link layer, an attacker may try to associate the media access control address (MAC) of a device he

[1] Not to be confused with the local area network, which refers rather to the geographical area such as a home, school or the premises of a company.

controls with the Internet Protocol address (IP) of another, genuine host in the network. For this, the address resolution protocol (ARP) is used, hence this type of attack is called ARP spoofing or ARP cache poisoning. Counterfeiting (spoofing) an IP-address for sending packets to a target host from a supposedly trustworthy source happens on the internet layer.
The majority of web-based attacks uses transport and application layer as attack vector. Attacks may be aimed at a webserver exploiting vulnerabilities in a server's configuration, scripts, databases, protocol versions, services or OS. From a server, a client may be attacked using vulnerabilities in the browser or other services that contact the server. An attack can also be initiated by a 'man in the middle', who controls the communication between server and client.

2.4.4 Injections

The goal of an injection attack is to trick the target system into treating a data input in an unauthorized way. A maliciously crafted user input, protocol message or file may exploit this type of vulnerability.
Examples are

- SQL-injections: Input to a database query can be manipulated to execute different SQL-Code than intended by the application.
- Lightweight Directory Access Protocol (LDAP) injection: Input is used to manipulate file system operations such as search, add, modify or delete in an unauthorized or unintended way.
- An operating system (OS) injection: An application passes user-controlled input directly to a call of a system function that will execute with the same authorization with which the application is run.
- Code injection using a buffer overflow as discussed in section 2.3.

Injection attacks can occur whenever input data from a potential untrustworthy source is processed by a computer program. So, an injection can happen anywhere, if the program's input is not properly 'sanitized'. Hence, proper input sanitization should always be subject to thorough testing.
Typical methods for input sanitization include

- Whitelisting: Only allow previously defined input patterns. Everything not on the whitelist is forbidden. Data handling is aborted when a pattern different from a whitelisted one is recognized
- Blacklisting: Reject predefined input patterns or forbid a set of specified inputs. Everything not on the blacklist is allowed, e.g. filter the data recursively by removing not allowed sequences or abort data handling when a forbidden pattern is recognized.
- Encode all input and output so that no undesired interpretation of an input is possible e.g. changing an executable command as an input to a simple string or transposing all characters to an upper-case format.

A specific method to prevent SQL-injections is the use of prepared statements, where the user's input is encoded and bound to string variables. These string variables are used in fixed (prepared) statements. The application then uses only this type of statement to query a database.

2.4.5 DOS and DDOS

“In a denial-of-service (DoS) attack, an attacker attempts to prevent legitimate users from accessing information or services. “[URL:US-CERT-DOS]
Denial of Service can be achieved by either crashing an application using a known bug or by overloading the target system’s processing capabilities. The latter can be achieved by triggering a resource eating (e.g. memory, CPU time) computation or by overloading the targets communication interfaces.
In a distributed denial of service attack (DDoS), single requests from a large number of computers or other digital devices connected to the Internet overload a target’s connection to the net or its resources.
DoS and DDoS attacks may also aim at any element that connects a targeted service with the internet.

2.4.6 Cross Site Scripting (XSS)

Cross Site Scripting (XSS) is a type of injection attack where a script is executed by a user’s browser that does not originate from the site actually visited.
There are three major types of XSS applied by attackers.

- In a persistent or stored XSS attack an attacker stores malicious scripts directly on a webserver. Such scripts may be hidden everywhere, where a user can enter data that is displayed to other users.
- Reflected or non-persistent XSS injects executable code in a query sent by the user. The executable code is reflected back to the user by the server, e.g. as part of an error message and then executed by the browser. A query containing the injection may involuntarily be sent by a user when clicking a crafted link in an e-mail or on another website.
- DOM based XSS exploits vulnerabilities in the implementation of the DOM-API or its use on the client side. DOM stands for the Document Object Model standard [URL:W3C-DOM]. It is implemented in different APIs that support dynamical website creation in browsers. When a script for dynamical website creation applies external input (e.g. from the user) to create a website’s representation dynamically in the browser, it may be vulnerable to this type of attack. An attacker can use a similar approach as in reflected XSS to trigger a DOM based XSS vulnerability. The main difference is, that the attack is staged locally in the user’s browser and may not even involve the webserver (except for providing a potentially vulnerable script).

A good defense against any form of XSS is a proper input sanitization on any user controlled input to a website. For further reference see [URL:OWASP-Categories].

2.4.7 Cross Site Request Forgery (CSRF)

“Cross Site Request Forgery (CSRF) attacks occur when a malicious web site causes a user’s web browser to perform an unwanted action on a trusted site.” [Zeller 08] As in reflected XSS, an attacker will trick the user into calling a linked address. However, the technical approach is different. In a CSRF, a query or request to a website comes from a seemingly trusted and authenticated source and is processed by the site. The call to the site may happen without the user noticing it, e.g. because he opened an E-Mail in an application that had the option ‘automatic display remote content’ switched on. A typical CSRF attack for example on a home router

[URL:router-hack] could be hidden in an E-Mail that will trigger an http request to the router: “http://admin:12345@192.168.1.1/start_apply.htm?dnsserver=66.66.66.66”. If the router password is set to the default password (in the example 12345) the attacker can now redirect any request for a website to wherever he wants it to go.

2.4.8 Web Vulnerabilities in Authentication and Authorization

The goal of exploiting authentication and authorization vulnerabilities usually is to steal information such as documents, credentials or financial data. Typical vulnerabilities include

- Leaks and flaws in custom built authentication and session management[1]. Authentication may be circumvented using injections that exploit weaknesses in the website’s script code.
- Insufficient access-checks to objects may be by-passed, e.g. if the application uses external input to construct a path to the requested resource.
- An application checks only if a user has authenticated, but not which resources he is allowed to access.
- Predictable cookies: Cookies are key value-pairs that are sent back and forth between client and server. Html is a stateless protocol. That means, if an already authenticated user sends more than one query via http or https, the server cannot decide by means of html only whether the user is allowed to do so. A unique cookie sent with a request, however, can tell the server that the user has authenticated and is allowed to request a resource. If a cookie is predictable by an attacker (e.g. because it is just the encoded user’s name) an attacker can simply spoof the user’s identity with no effort. If a cookie is stolen, it might be used in the same manner – especially if the cookie never expires.

2.4.9 Encrypted communication and website authenticity

A good modern encryption system implements the following principles:

- Kerckhoffs’s principle: The strength of encryption is independent of the secrecy of the method used to encrypt a message. It should just depend on number of available keys and the time an attacker needs to find the right key.
- The cypher text should appear random, so that statistical analysis will not break the cypher.
- If only a part of the message changes, the whole cipher text – not only a part must change. Else a differential analysis[2] may break the cipher.
- Perfect forward secrecy [MIT17]: Each message exchange uses a different key. Even if the cipher of an information exchange is broken in the future, other intercepted cipher texts remain secret.

Symmetric encryption

The classical way of encrypting a message uses a key together with an algorithm to transform a clear text into a cipher text. The backwards transformation uses the same key and an inverse algorithm to decipher the encrypted message. Given the key and the algorithm, encryption and decryption are fairly straightforward. Well-known historical examples are Caesar and the Vigenère cypher. Another well-known

[1] A ruleset implemented to manage the interactions e.g. between a client and a server or a user and a web application during a session. A session is established by a login and finished by a logout.

[2] If an attacker intercepts two similar message, a first message and one with a small correction, e.g. the date, he can compare the differences in the ciphertext.

example is the one-time pad. This method uses a random different key for each message, at minimum as long as the message itself, which must never be repeated. If an attacker cannot get hold of or guess the key, the onetime pad is provably unbreakable. The weakness of any symmetric method is the need to exchange keys for encryption and decryption beforehand in a secure manner.

Asymmetric encryption

The name asymmetric encryption is owed to the fact that two different sets of keys are used. Each user creates a pair of keys: The first key of the pair is published and therefore called 'public key'. It can be used by anybody who wants to send an encrypted message to the key-owner. The second key is the 'private key' and used for decryption by the key-owner. Hence, this key must be kept secret under any circumstances. The methods in use are based on complex mathematical concepts. Asymmetric encryption uses mathematical functions that can fast and easily calculate the clear text from a cypher, when the proper private key is known but take a huge amount of computing power (preferably several hundred thousand years) if this is not the case. The method does not require a secure key exchange. Its secrecy depends on the amount of time required to guess or calculate the private key, and on the assumption that no algorithmic approach is found, or no computer is built, that is able to speed up that process by several orders.

Website authenticity

Asymmetric encryption is also used for electronic signatures. Trusted Certificate Authorities issue digital certificates. A digital certificate certifies the ownership of a public key by the named subject of the certificate. The certificate authority is an organization that ensures the authenticity of the certificate holder and electronically signs the issued certificate. Browser and operating system vendors rely on the certificates issued by such organizations. Many certificates are so-called domain validated certificates that do not assure that any particular legal entity is connected to a website's certificate. An Extended Validation Certificate is issued if the legal identity of the website owner has been established and confirmed. This type of certificate should be used e.g. for a web shop.
A self-signed certificate has been signed using the secret key of its issuer and can be verified by its own public key. This means that no third party such as a Trusted Certificate Authority is involved in establishing the trustworthiness of the certificate. Self-signed certificates should be considered as untrustworthy.

2.5 Social Engineering (20 minutes)

Social engineering in the context of information security is any psychological manipulation of human beings applied to gain information about or access to computer systems.

A typical social engineering attack cycle [URL:SEF] consists of 4 phases:

1. Information gathering: Collect information on the target, e.g. employees, platform users, potential victims etc. Typically social media, company or private websites, etc. are often used as information sources.
2. Establish relationship and build trust: Depending on the target, this can be done by repeated phone calls, e-mails, contact via social networks etc.

3. Exploit: The attacker uses gathered information and established relationships to infiltrate the target.
4. Execution: The ultimate attack goal is reached and tracks are covered if possible, so that the victim may not even notice that an attack has occurred.

Common forms of social engineering are

- Phishing: An e-mail that appears to be sent from a reputable source tricks the user into visiting a malicious website, providing personal information or downloading malicious software. A special form is spear-phishing, which means the phishing mail is personally adapted to the victim or a smaller group of people. Another special form is called 'whaling' and is a form of spear-phishing aimed at a company's top management.
- SMiShing: Same as phishing, but using SMS, MMS or other messenger services as medium.
- Vishing: The practice to elicit information or exert influence on a person via a phone call. Attackers often impersonate customers, technical support of phone companies or Internet service providers or employees of financial institutions.

2.6 Security in Wireless Communications (25 minutes)

In a wireless communication information is transferred from a sender to a receiver, where both are not connected by an information carrying device such as a conducting cable or an optical fibre. As a consequence, the transmission of information is accessible to anybody with suitable technical means.

Threats to wireless communications

An attacker might set up a receiver and record transmissions. This is called sniffing. It is a passive form of attack. In the case of a wireless transmission, it usually cannot be detected. Sniffing is also applied in non-wireless networks like computer networks.
The attacker can set up a sender and try to disrupt a transmission by sending interfering signals (DoS) or pose as a legitimate information source (spoofing).
If information is exchanged over relay stations such as radio repeaters or base stations in mobile communication networks, or a wireless router in a computer network, the attacker might stage a Man-in-the-Middle attack by mimicking the relay. Especially a relay station that offers an access point to the internet, is often used in these attack-types.

Wireless remote controls and other devices receiving commands over a wireless channel may be vulnerable to a 'replay attack'[1]: An attacker records a device's signal and replays it later using a suitable radio unit. A garage door or a car might be opened by a burglar applying this technique to an insecure system.

Please note, that these potential threats to wireless communication systems are also relevant to non-wireless systems.

Threat mitigation in wireless communications

To secure communications over a wireless channel requires at least all of the following measures:

[1] Replay attacks generally work by replaying messages from an authorized source. This type of attack is also performed in computer networks to fool weak authorization mechanisms.

- All communications must be adequately encrypted.
- All senders and receivers must have tamper-proof identifiers.
- A secure authentication process for senders and receivers must be in place.

An appropriate encryption mitigates sniffing and the preparation of spoofing and man-in-the-middle attacks. Tamper-proof identifiers together with adequate authentication mitigates spoofing and man-in-the-middle attacks.

WiFi is a short name for Wireless Fidelity and describes technologies for establishing connections within a WLAN (Wireless Large Area Network) that conform to the IEEE 802.11 standard (Vijay K. Varma on IEEE Emerging Technology portal, 2006 - 2012) and thereby offers access points to a local net or even the internet.
Communication is established using electromagnetic waves in the UHF and SHF[1] radio bands.
Open WiFi-stations fulfill none of the above requirements for a secure communication. Identifiers such as a MAC-address (media access control address) of a device or an SSID (Service Set identifier, i.e. the 'name' of the wireless network) can be spoofed. Communication between a device and the WiFi station is not encrypted and there is no authentication process. To secure WiFi communications WiFi Protected Access protocols WPA2 and WPA3 must be used. These protocols support additional identification and authentication methods such as using a password, a username/password-combination or a digital certificate. Communication between a device and an access point is adequately encrypted.
Bluetooth is a wireless technology standard maintained by [Bluetooth SIG] for exchanging data directly between devices over short distances[2].
An easy to use identification and authorization mechanism (device pairing) is in place and communication is encrypted.
Flaws in the protocol or its use or poor protocol implementation may cause vulnerabilities in Bluetooth as well as in WiFi communications.

[1] UHF = Ultra High Frequency from 300 MHz up to 3 GHz, SHF from 3GHz to 40 GHz
These bands are also used by some remote controls, e.g. presenters, car keys or garage doors. Bluetooth also uses UHF-frequencies. In the UHF spectrum, WiFi and Bluethooth typically use the open ISM-band between 2,4 and 2,48 GHz, which is reserved for industrial, scientific and medical radio traffic.
[2] Depending on the Bluetooth class of a device, this may range between half a meter and 100 meters.

3. Security in the Software Lifecycle (350 minutes)

Terms

Application Security Development Lifecycle Process (ASDL), CVE, CVSS, DREAD, data flow diagram (DFD), design pattern, design principle, ethical hacking, fuzz testing, incident response plan, misuse case, penetration test, security framework, security plan, STRIDE, threat modelling, vulnerability assessment

Learning Objectives

(3.1.1) Describe major activities in the Security Development Lifecycle Process (K2)

(3.2.1.1) Know the relations between threat, requirement and mitigation (K1)

(3.2.1.2) Allocate potential threats in a given dataflow diagram of a system and suggest activities that are suitable to mitigate the risk involved. (K3)

(3.2.1.3) Interpret threats and threat mitigations represented in a given misuse case diagram. (K3)

(3.2.2.1) Relate STRIDE categories to a given vulnerability. (K3)

(3.2.2.2) Explain the purpose of DREAD and each of its five categories. (K2)

(3.3.1.1) Describe the purpose and goal of secure design principles (K2)

(3.3.1.2) Know the purpose and goal of secure design patterns (K1)

(3.3.2.1) Describe purpose and goal of secure coding (K2)

(3.4.1) Recall typical goals of security testing. (K1)

(3.4.2) Name typical entry and exit criteria for security testing. (K1)

(3.4.3) Differentiate between different types of security testing with respect to purpose and goal. (K2)

(3.5.1) Recall why security relevant defects should be treated differently from other defect types. (K1)

(3.5.2) Recall how the MITRE Common Vulnerabilities and Exposures (CVE) initiative works. (K1)

3.1 The Security Development Lifecycle Process (30 minutes)

Howard and Lipner state in [Howard 06]: *"Present software development methods lack in-depth security awareness, discipline, best practice, and rigor…"*. They describe an approach adopted by Microsoft that considers system security in every phase of the software lifecycle.
An Application Security Development Lifecycle Process (ASDL) introduces security activities in all steps required to develop, manage, operate and maintain an application. It also covers infrastructure management as well as audit activities. A generalized process is discussed in [ISO/IEC 27034].
The ASDL is not a sequential model. It describes which process phases and disciplines must be tailored into an existing lifecycle. Although the cited standard focuses on application security, the phases apply as well to system development and systems engineering[1]. A lifecycle-model that ensures an adequate level of security requires at least (e.g. [ISO/IEC 27034] and [Howard 06]):

1. Training: All personnel involved (e.g. developers, testers and program managers) in building an application undergo basic security training. Topics for basic training are privacy, threat modeling, secure design, coding and security testing. Advanced trainings on these topics are added as required.
2. Requirements include application security requirements as well as process requirements. The latter incorporate when and how often threat assessments are executed or what type of security testing is required in different life cycle activities. In this phase, the project team must also define item pass/fail criteria for security related defects that will be monitored throughout the project[2].
3. Design: The application design phase requires the identification of attack surfaces and threat modeling (see section 3.2 of this syllabus). The purpose of these activities is to uncover security flaws in the application's architecture and to identify where security mechanisms must be implemented.
4. Implementation activities enforce the use of approved tools for static analysis during coding. Unsecure functions are discovered and banned. This ensures that known weaknesses remain in the code and prevents insertion of new weaknesses into the code.
5. Verification: All verification phases include specific security test activities, such as dynamic analysis and fuzz testing (see section 3.4.2). Verification has to include risk management, at least a review of all attack surfaces. The review ensures that the product is suitably protected, and no new attack vectors have been uncovered by testing.
6. Release activities in this context are the actions required to prepare a release. They include a final review of all security activities and their success. It ensures that the ASDL process has been followed correctly throughout all development phases. In an agile environment it can be done incrementally for each sprint. Before an application is released, the responsible project team must also define an incident response plan for the time after release. The plan describes required actions on who must be informed, how to analyze the incident and to mitigate or eventually eliminate the corresponding threat. Finally, the release configuration should be archived in a way that allows

[1] See [NIST-SP-800-160] for comparison.
[2] See ‚bug bars' in the Microsoft Security Development Process [Howard 06].

detecting alterations by third parties during or after shipment (e.g. download). A common way to do this is 'signing' a release using a cryptographic hash [URL:Signing].

7. Main Response activities involve establishing a response team and implementing the response plan for the released product.

A security plan after [NIST 800-39] can describe how these activities are implemented for a given project. The security plan may refer to a test plan for testing activities and vice versa. In agile approaches, the activities must be reflected e.g. in team charters, definitions of ready, definitions of done and acceptance criteria.
A security plan should also be compliant to the organization's security framework. A security framework is a set of inputs, desired outcomes, policies, activities and roles to identify, detect, protect against and respond to threats for an organization's assets. It should also include recovery measures [NIST 2018].

3.2 Threat modeling & Requirements Engineering (145 minutes)

Threat models are used to identify security requirements, requirements for mitigation and non-requirements, if a threat cannot be mitigated (see chapter 12 in [Shostack 14]). They are also used to assess mitigative actions, as well as to identify and prioritize test conditions for security tests. There is a strong interplay between assessing threats, identifying possible mitigations and defining requirements [Shostack 14]. Therefore, these tasks have to be performed iteratively.

Threat modeling is any structured approach that identifies, rates and manages security risks associated with a system or application. It is a strictly risk based approach and incorporates the steps

- Decompose the system and identify attack surfaces and corresponding threats
- Determine and rate the threats
- Mitigate

The methods presented in this section are typically applied in the ASDL requirements and design phase.
Guidance on how to identify and analyze threats and devise appropriate counter measures on an organizational level can be found in [ISTQB-AL-SEC] and in [NIST-SP-800-37]. [Shostack 14] describes a comprehensive approach to threat modeling, defining security requirements and implementing preventive and mitigative actions.

3.2.1 Threat identification

Use case diagrams help to identify threat agents and attack vectors in early design phases. To this end existing use cases are extended by a description of a behavior that the system/entity owner does not want to occur [Sindre 05]. The latter is called a misuse-case. In addition the set of actors from the original use cases is extended by mis-actors. A mis-actor is the opposite of an actor and the system is not supposed to support that type of actor.
The derivation of misuse cases from use cases is accomplished in 5 steps [Sindre 05]:

1) Describe the normal actors and use cases

2) Identify major mis-actors and misuse cases
3) Investigate and model the relations between use cases and misuse cases.
4) Introduce additional use cases that detect, prevent or mitigate misuse cases.
5) Elicit and document detailed security requirements from the revised use case models.

From a requirements and development perspective, misuse cases support the identification of required security functions.
From the test perspective, misuse cases from step 2 and use cases from step 4 represent test conditions. Both can be used to develop penetration test scenarios.

Data flow diagrams (DFD) support threat modeling on a system architecture level.
A data flow diagram incorporates [DeMarco 78], [Howard 06]

- Data storage facilities – represented by two horizontal, parallel lines.
- Entities delivering or receiving input or output – depicted by rectangles.
- Functions or (sub-)systems that process data and data flows – shown as circles in a DFD.
- Data flows between the components above – modeled using continuous lines with an arrow tip
- System boundaries – indicated by dotted lines [Pohl 11]

DFDs help to identify which data flows may be tainted and where this could cause harm. Software architects and developers can derive from these diagrams where security mechanisms must be provided. A tester can derive which external and internal system interfaces must undergo security testing.

Threat modeling in later (e.g. detailed) design and implementation phases requires a deep technical understanding of data structures, memory layouts, programming languages and communication protocols in use. This requires the skills of a dedicated hacker and is excluded from here

3.2.2 Threat determination and rating

Once attack surfaces and related threats have been identified, their relation should be further analyzed, classified and rated.
To support this, Microsoft adopted the STRIDE methodology ([URL:STRIDE], [Howard 06], [Howard 02]). For each identified threat one of the following six categories is determined:

- S – Spoofing Identity: Any threat action aimed at attaining another entity's authentication information and using it. An example for a threat in this category would be a user interface with unsecure certification mechanisms.
- T – Tampering may occur when an attacker could modify or change persistent data. Stored XSS and installation of malware are examples that fall in this category.
- R – Repudiation: Malicious actions or system weaknesses that prevent a later analysis. Especially missing functions to trace legal as well as illegal transactions make repudiation possible.
- I – Information Disclosure: An attacker can read information without the required level of access. A specific example is the already mentioned Heartbleed-vulnerability.
- D – Denial of Service: See chapter 1 for an explanation.

- E – Elevation of Privilege: This category applies if a system or application allows actions aimed at elevating the authorized access level to one that is entitled to privileged information or services, e.g. admin rights.

DREAD [Howard 02] considers 5 risk factors. For a given threat, each factor can be rated on a scale reaching from low (1) over medium (2) to high (3). The values are added up to estimate the overall risk rating for the threat. A value ranging from 12 to 15 indicates a high risk, values ranging from 8 to 11 a medium risk and all lower values (5 to 7) a low risk (see [URL:DREAD] as reference).
The factors in DREAD translate to

- D – Damage potential. Potential damage always depends on an application's or system's context. Accordingly, each project must define its own rules for determining the damage level.
- R – Reproducibility. A high reproducibility (3) means that the attack requires no specific time window. Reproducibility is medium (2) if the attack is only possible at specific times, e.g. during a backup. If the attack is very difficult to perform, even with knowledge of a corresponding vulnerability, reproducibility is set to low (1)
- E – Exploitability rates the skill level required for executing the attack. High (3) exploitability is assumed if a script kiddie could perform the attack. If an exploit requires a dedicated hacker, exploitability is medium (2). It is low (1) if only a skilled hacker can perform the attack.
- A – Affected users. Here numbers of affected user and typical vs. untypical configurations have to be considered. A high rating (3) is suggested if all or key users are threatened under default configurations. Medium (3) is applies to non-default configurations or when only some users are threatened. Low is used when the weakness is effective only for uncommon features or configurations.
- D – Discoverability for the attacker. Discoverability must be rated high (3) if the corresponding vulnerability and its potential malicious use is common knowledge or could easily become so. Since this can be the case with any vulnerability, it is good practice to rate this factor either as high or low.
  Low discoverability (1) means that an attacker would need an intimate knowledge of the internal workings of the system or application[1].

DREAD often leads to subjective results and its use is not any longer recommended by Microsoft [Shostack 14]. Nevertheless, it is easy to use and contains all major factors that are required for rating the severity of a vulnerability.

[1] Please note that discoverability is not the same as exploitability. For example: An apt C programmer might have difficulties to spot a weakness in a Java code. But once pointed at it and given some time, he will very probably be able to write an exploit.

3.3 Secure Design and Secure Coding Principles (95 minutes)

3.3.1 Secure Design

Identifying and applying suitable secure design techniques is part of the third group of ASDL-activities. Here secure design principles and secure design patterns should be applied.

Design principles

Design principles are guidelines that help to avoid common design flaws, such as the possibility to circumvent an existing security feature.
At least the following principles should be considered[1]:

- Complete mediation: Every access to every object must be checked for authorization.
- Least privilege: Only give the privileges required to perform a task for the required time period. Never more. Never longer.
- Need-to-know: Access to data should not be broader than what is required for legitimate purposes.
- Auditability and recovery: Events must be logged so that attacks can be analyzed. Data and services must be protected against loss and denial of service.
- Secure the weakest link: Reduce attack surface in an efficient manner. Start building defenses for undefended access points to your system, before improving existing defenses.
- Defend-in-depth-and-diversity: Apply several layers of physical, technical and administrative controls to defend a system
- No security by obscurity: Do not rely on the secrecy of your design or implementation as only method of providing security. This will fail eventually.
- Secure defaults: The default configuration of a software or system should correspond to the maximum security settings for the system.
- Fail in a secure manner: Programs have bugs and are prone to operating errors. Identify and mitigate according security risks so that even if the system or parts of it fail, a defined level of security can still be maintained.
- Input validation: All input should be validated by a system, before it is used.
- Output encoding: Prevent that the output from one system could be used for an injection attack on another system.
- Separate code and data: Data should never be allowed to be misinterpreted as executable code[2].
- Usable security: Support users in making secure choices when working with the system. Try to design the system in a way, that its main uses and security needs do not collide.

[1] See also section 1.3.
[2] The same is of course valid for a mix-up of user-input and configuration data, as e.g. in some CSS-injections – see e.g. https://www.owasp.org/index.php/Testing_for_CSS_Injection_(OTG-CLIENT-005).

Design patterns

A design pattern is a general reusable solution to a common design problem. Secure design patterns aim at avoiding or mitigating the accidental insertion of common vulnerabilities into a system.
Existing certified implementations for common design patterns should (if available) be used for

- Identification and authentication
- Authorization, e.g. in session management and access control.
- Encryption

Without proper expertise and due diligence, self-designed solutions may incorporate severe design flaws and resulting vulnerabilities in deployed products and services.

3.3.2 Secure Coding

Secure coding rules aim at avoiding typical security relevant programming mistakes that may lead to

- Insecure handling of inputs and outputs
- Insecure handling of storage and memory, that may for instance lead to buffer overflows
- Insecure handling of race conditions that may cause crashes, unreliable checks for authorization like wrong order of check for authorization and access to data
- Insecure handling of permission and ressources, e.g. unspecified default permissions

An organization should implement rules for secure coding that incorporate

- General secure design principles and patterns (see section 3.3.1 for examples).
- Programming language specific rules[1] to avoid the use of insecure language constructs and language specific mistakes. [URL:CERT SEI] gives examples for C and C++.
- Regular and compulsory use of static analysis and code reviews (see also section 3.4) to enforce adherence to the rules.

3.4 Security Testing (60 minutes)

This syllabus focuses on major security objectives as well as a practical understanding of threat assessments.
An in-depth discussion of security testing processes, a detailed view on security testing goals and an organization wide security risk management process is discussed in [ISTQB CTAL-SEC].

Typical goals for security testing are:

- Increase security awareness and create a business case for security measures by showing the possible outcomes of lack of security.
- Identify gaps in security in system architectures, designs, operations and organizational policies and processes.

[1] Including compiler settings

- Suggest improvements to existing security mechanisms.
- Meet regulatory compliance.
- Discover new threats.
- Discover weaknesses in programs, systems and procedures.

3.4.1 Security testing objectives, entry- and exit-criteria

Test objectives

Three major factors govern the specification of security testing objectives ([Palmer 01], [Ruef 07]):

- The asset that must be protected.
- Against what the asset must be protected
- The budget available for testing and improving the asset protection.

In order to develop verifiable security testing objectives, a detailed description of the assets and related systems and services is required. General assertions as 'the database', 'the web shop services' will not do (([Palmer 01], [Ruef 07]).
For identifying and describing against what to protect, a description of threat agents and attack vector e.g. in terms of CIA (see chapter 1) or STRIDE (see section 3.2) is required. Security tests require a suitable budget: A budget that is too low, may only permit a conceptual audit (see below), where a full-fledged penetration test is required. A risk based approach that applies for example STRIDE and DREAD will help to focus the budget.

Entry and exit criteria

Certain criteria must be met before any security testing may start.
The minimum entry criteria for starting any security testing are:

- A permission and mandate to perform the tests as described in a security testing plan[1]. For penetration tests, the permission must come from an authorized body in a legitimate form. Sponsor and penetration testers must agree on it. This type of permission is often referred to as the 'get out of jail-free-card' in allusion to a well-known cardboard game [Palmer 01].
- The test objectives, target systems and the test scope are clearly defined.
- Accountable contact persons for each target system in the organization are identified, involved and available.
- An appropriate vulnerability management including vulnerability categories is defined.

Exit criteria ensure that all security-testing objectives are met. In an ASDL, exit criteria should be defined for each phase in terms of test item coverage and item pass/fail criteria.
Coverage criteria may include

- which interfaces must be covered with fuzz tests,
- where a penetration test on system test level must have been performed,
- which regression tests from prior releases must have been executed

Pass/fail criteria may determine

- which types of security issues must be fixed before release.
- which types of privacy issues must be fixed before release

[1] Contents of such a plan are described for instance in [CT AL-SEC]

Penetration test results often include suggestions how to fix a discovered vulnerability. This is similar to many types of efficiency testing that aim at performance optimization by suggesting performance improvements.

3.4.2 Security Testing Types

Security audit

Organizational security audits ([RUEF 07], [ISTQB CTAL-SEC]) evaluate an organizations processes and procedures with respect to a set of defined security policies and goals. In the ASDL organizational audits should be part of the requirements phase. A typical review object is the security plan. The audit must ensure that all development phases include adequate activities for building a secure product. An organizational audit of a software project should be repeated in the release phase as part of the final security review.
Conceptual audits [RUEF 07] assess if a proposed or existing solution offers an adequate level of security. Conceptual audits can be executed independently or as part of a vulnerability assessment. In the ASDL conceptual audits should be part of the design phase and should be repeated on a regular basis in implementation, verification and release. For an already existing product they should be integrated into the change management process.
Organizational and conceptual audits are reviews, i.e. static tests.

Vulnerability assessment

A technical vulnerability assessment ([RUEF 07], [Weidman 14]) aims at detecting vulnerabilities as economically as possible. It may involve conceptual audits and static analysis, as well as dynamic techniques. Dynamic tests may include vulnerability scans, fuzz testing and even full-scale penetration testing. The techniques actually applied depend on the context of testing [ISTQB_CTFL] and concentrate on specific goals.
Vulnerability assessments should be an integral part of any SDLC, starting at the latest in the design phase.
Vulnerability assessments are also a common task in maintenance testing – see [ISTQB_CTFL].

Static analysis

A static analysis evaluates a software development artifact without execution, typically with the help of a tool.
Static analysis in developing secure systems and applications can detect insecure functions used by a programmer. Examples for analyzers that can detect such constructs are the clang-analyzer for C/C++ and FindBugs [URL:FindBugs] and PMD [URL:PMD] for Java. A common approach is the taint flow analysis: The static analyzer transforms the source code into a model of the software and locates areas in the software that may be vulnerable to injections from the outside. Taint flow analysis uses techniques that are similar to those applied for data flow analysis (see [ISTQB CTAL-TTA]). A tool example that uses static taint-flow analysis for php-code is [URL:RIPS].
Static analysis should be applied routinely during any software and system development activities.

Dynamic analysis

In security testing dynamic analysis is performed in different contexts. It is applied to monitor code as it executes, e.g. in dynamic taint analysis and dynamic symbolic execution [Schwartz10]. Dynamic taint analysis has the same goal as static taint analysis. It can detect vulnerability against injections in an application or system. Another form of dynamic analysis is vulnerability scanning. Vulnerability scans check computers, computerized systems, networks or applications for known or potential weaknesses. A vulnerability scanner may discover unpatched and hence insecure program or OS versions, insecure configurations such as file or access permission settings, turned off authentication mechanisms and many more. Tools in use range from network scanners to penetration testing frameworks. Both may possess large databases of executable exploits for different targets, which can be run automatically to uncover known vulnerabilities. Dynamic analysis should be part of verification, release and response activities, as well as maintenance testing.

Fuzz testing

Fuzz testing verifies an application's input validation capabilities. In a fuzz test, malformed inputs are created with the help of a tool. These inputs are fed to the application or system under test, to see how it reacts. If the application fails, a potentially security relevant bug has been found. [Howard 06] names three categories of fuzz testing:

- File format parsers that manipulate document, executable and image file types.
- Network protocol parsers that manipulate data package contents and package sequences.
- Application Programming Interface (API) parsers and miscellaneous parsers for specific API's or purposes.

In 'dumb fuzzing' the input data is changed at random. In 'smart fuzzing' specific relevant parts of a data structure are changed. Test design techniques like equivalence partitioning and boundary value analysis support smart fuzzing approaches.

Penetration testing and Ethical hacking

Penetration testing (penetration testing) refers to any *"test methodology intended to circumvent the security function of a system"* [NIST-SP-800-160]. As such, it may be understood as an umbrella term for all types of testing mentioned above. The term is often used as a synonym for ethical hacking. Ethical or white hat hacking is widely understood as performing an authorized attack on an existing computer system or network to assess its security [Palmer 01]. White hat hacking may apply all testing types mentioned above as well as social engineering techniques to break into a system. The typical ethical hacking approach is to take the view of a real malicious attacker and demonstrate what the attacker could do – without anybody noticing the attack. This syllabus sees penetration testing in this meaning – as a proof of concept that a specific real attack is possible.

Security testing in its different forms should be applied throughout the software lifecycle. In the ASDL penetration tests may be performed during release and response. In fact, many companies run a bug bounty program[1] in alignment with their incident response plan.

[1] A program that offers individuals recognition and maybe money for reporting bugs.

3.5 Defect management and classification (20 minutes)

Defect management supports software security management throughout the lifecycle of a product or service. Defect management for security related defects should be set up early in the requirements phase of the ASDL. Security relevant defects require special access rights. Only authorized and trusted personal should have admission to read how an existing vulnerability can be exploited. Such details should also never be included in a general test report [Palmer 01]. Also a security defect classification scheme should be adopted early on. Examples for classifications in use are CIA, STRIDE, DREAD or different versions of the Common Vulnerability Scoring System CVSS [NISTIR 7946]. This supports risk management and required re-assessments of security and organizational audits.

Incident management also plays a crucial role in the response phase of the ASDL. It is a real challenge to inform customers about vulnerabilities. Due to this, organizations like Apple, Microsoft, Red Hat and many others publish vulnerabilities and their remedies via the MITRE CVE list. The MITRE corporation is a not-for-profit company that operates multiple research and development centers [URL:MITRE] in the United States. CVE is short for 'Common Vulnerabilities and Exposures'. The CVE list assigns a unique identifier to each posted vulnerability.
The process begins with the discovery of a potential security vulnerability [URL:CVE-FAQ]. The next step is to request a CVE identifier (CVE ID) from a CVE numbering authority (CNA). A CNA is an organization[1] authorized by MITRE to assign an ID to a vulnerability and publish it on the CVE list. A complete list of CNAs is on [URL:CNA]. Certain issues require a software manufacturer to contact a specific CNA. The current list can be found under [URL:CVE-List].
The National Institute of Standards and Technology (NIST) also maintains a national vulnerability database [URL:NVD] that contains all CVE list entries.

[1] Including MITRE

4. References

4.1 ISTQB Documents

[ISTQB_CTFL] ISTQB Foundation Level Syllabus, Version 2011

[ISTQB CTAL-TTA] Certified Tester Advanced Level Technical Test Analyst, Version 2012

[ISTQB CTAL-SEC] Certified Tester Advanced Level Syllabus Security Tester, Version 2016

4.2 Standards

[CC-PART-1] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model, Version 3.1, Revision 4, September 2012

[CC-PART-2] Common Criteria for Information Technology Security Evaluation Part 2: Security functional components, Version 3.1, Revision 4, September 2012

[CC-PART-3] Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components, Version 3.1, Revision 4, September 2012

[ISO 25010] Systems and software engineering –
Systems and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models

[ISO/IEC 27001] Information technology — Security techniques — Information security management systems — Requirements, 2013.

[ISO/IEC 27034-1] ISO/IEC 27034-1: Information technology – Security techniques – Application security – Part 1: Overview and concepts

[ISO 31000:2018] ISO 31000:2018, Risk management – Guidelines

[NIST 2018] Framework for Improving Critical Infrastructure Cybersecurity. Version 1.1, NIST, 2018

[NIST-SP-800-37] NIST Special Publication 800-37: Guide for Applying the Risk Management Framework to Federal Information Systems, Rev. 1, 2010, updated 2014.

[NIST-SP-800-39] NIST Special Publication 800-39: Managing Information Security Risk, Organization, Mission, and Information System View, March 2011

[NIST-SP-800-160] NIST Special Publication 800-160 Systems Security Engineering, NIST, November 2016

[RFC 1122] Requirements for Internet Hosts -- Communication Layers, url: https://tools.ietf.org/html/rfc1122, last retrieved July 14, 2017

[RFC 1123] Requirements for Internet Hosts -- Application and Support, url: https://tools.ietf.org/html/rfc1123, last retrieved July 14, 2017

4.3 Books

[DeMarco 78] Tom DeMarco, Structured Analysis and System Specification, Yourdon Inc., New York, 1978

[Eck 14] Claudia Eckert, IT-Sicherheit – Konzepte – Verfahren – Protokolle, Oldenbourg Verlag, 9th edition, 2014

[Howard 02] Michael Howard and David LeBlanc, Writing Secure Code, Microsoft Press, 2002

[Howard 06] Michael Howard and Steve Lipner, The Security Development Lifecycle, Microsoft Press, 2006

[MIT03] Kevin Mitnick, William L. Simon, The Art of Deception: Controlling the Human Element of Security, Wiley Publishing Inc, 2003

[MIT17] Kevin Mitnick, The Art of Invisibility, Little, Brown and Company, New York, Boston, London, 2017

[Pohl 11] Klaus Pohl & Chris Rupp, Requirements Engineering Fundamentals, Rocky Nook Computing, 2nd edition, April 2015

[Ruef 07] Marc Ruef, Die Kunst des Penetration Testing – Handbuch für professionelle Hacker, C&L Computer und Literatur Verlag, Böblingen, 2007

[Shostack 14] Adam Shostack, Threat Modeling, John Wiley & Sons, 2014.

[Weidman 14] Weidman, G., Penetration Testing. A Hands-On Introduction To Hacking, No Starch Press, 2014.

[Whit03] James A. Whittaker, Howard H. Thompson, How to break Software Security, Addison Wesley, 2003

4.4 Other References (Articles and Web-Resources)

[Bluetooth SIG] Website of the Bluetooth Special Interest Group, https://www.bluetooth.com/ last retrieved on June 25, 2019.

[ENISA] ENISA Glossary, URL: https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/glossary#G51, last retrieved on July 11, 2017.

[Lamport 78] Leslie Lamport, The implementation of reliable distributed multiprocess systems, Computer Networks, Volume 2, Issue 2, Pages 95-114, Elsevier, May 1978.

[NISTIR-SP-800-53] Security and Privacy Controls for Federal Information Systems

and Organizations, NIST Special Publication 800-53, Revision 4, 2013 with updates from 2015. See http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

[NISTIR 7946] Joshua Franklin and Harold Booth, CVSS Implementation Guidance, NISTIR 7946, NIST, April 2014

[**Palmer 01]** C.C. Palmer, Ethical hacking, IBM Systems Journal, Vol. 40, No. 3, 2001

[Saltzer 75] Saltzer, Jerome H. & Schroeder, Michael D.: The Protection of Information in Computer Systems., Proceedings of the IEEE 63, September 1975

[Schwartz 10] Schwartz et al., All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask), SP '10 Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp. 317-331, May 2010

[Sindre 05] Sindre and Opdahl, Eliciting Security Requirements with Misuse Cases, Requirements Engineering, 10: 34 - 44, 2005

[URL:CAPEC] Common Attack Pattern Enumeration and Classification — CAPEC™, https://capec.mitre.org/index.html, retrieved on July 14, 2019.

[URL:CERT SEI] SEI CERT Coding Standards, https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards, retrieved on July 11, 2019

[URL:CISA alerts] Cybersecurity and Infrastructure Security Agency (CISA), Alerts, https://www.us-cert.gov/ncas/alerts, retrieved on June 25, 2019.

[URL:CNA] https://cve.mitre.org/cve/request_id.html, retrieved on August 3, 2017

[URL:CVE-FAQ] https://cve.mitre.org/about/#how_cve_works, retrieved on August 3, 2017

[URL:CVE-List] https://cve.mitre.org/cve/ , retrieved on August 3, 2017

[URL:DREAD] https://msdn.microsoft.com/en-us/library/ff648644.aspx, last retrieved on August 3, 2017

[URL:FindBugs] http://findbugs.sourceforge.net/, last retrieved on July 31, 2017

[URL:MITRE] https://www.mitre.org/, last retrieved on August 3, 2017

[URL:MSDL] https://www.microsoft.com/en-us/sdl, last retrieved on June 24, 2019

[URL:OWASP-Categories] https://www.owasp.org/index.php/Category:Attack,

[URL:PMD] http://pmd.sourceforge.net/pmd-5.2.3/, last retrieved on July 31, 2017

[URL:RIPS] http://rips-scanner.sourceforge.net/, last retrieved on July 31, 2017

[URL:router-hack] http://www.heise.de/security/meldung/Mail-hackt-Router-17**,** last retrieved on July 18, 2014

[URL:SEF] https://www.social-engineer.org/framework/general-discussion/, last retrieved on July 14, 2017

[URL:STRIDE] https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx, retrieved on August 2, 2017

[URL:Signing] Introduction to Code Signing, Microsoft Developer Network, https://msdn.microsoft.com/en-us/library/ms537361(v=vs.85).aspx , last retrieved on July 31, 2017

[URL:US-CERT-DoS] https://www.us-cert.gov/ncas/tips/ST04-015, last retrieved on July 14, 2017

[Zeller 08] William Zeller, Edward W. Felten: Cross-Site Request Forgeries: Exploitation and Prevention, Revision 10/15/2008, Princeton University, 2008, last retrieved July 23, 2017.