IT-Sicherheit

Klausur hat 100 Punkte

Was habt ihr aus der IT Sicherheit Vorlesung mitgenommen?

Dass IT-Sicherheit wichtig ist und worauf in der Entwicklung sowie dem Aufbau von IT-Systemen geachtet werden muss. Besonders die Grundprinzipien werden mich in meiner weiteren Karriere hilfreich begleiten.

IT-Sicherheit ist ein ständiger Prozess und sollte in den Development Cycle integriert werden. OWASP und andere Organisationen bieten hilfreiche Informationen und klären auf über Schutzmöglichkeiten.

IT-Sicherheit basierend auf Risikobewertung implementieren. Common Criteria beachten.

Es gibt Tradeoffs im CIA Triad die abgewägt werden müssen.

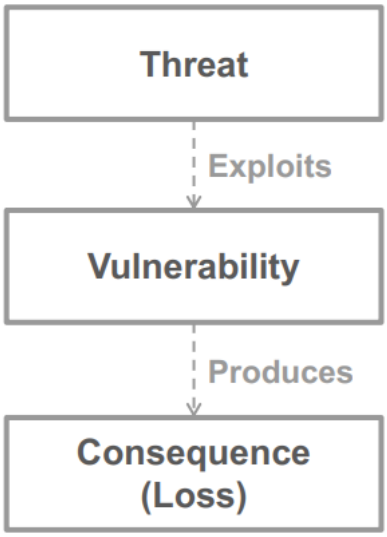
Ein ISMS zu nutzen ist wichtig und richtig!

Security Patterns haben (immer) ihre Daseinsberechtigung und sind wichtig. Ebenso wichtig ist das Minimalprinzip --> Je simpler der Source Code, desto geringer ist die Angriffsfläche.

Grundlagen

“Es ist nicht die Frage, ob man angegriffen wird, sondern wann”

“Sicherheit und Benutzerfreundlichkeit sollten integrale Bestandteile von Systemen sein und sich nicht gegenseitig ausschließen”



Threat: Ein Akt, der zu Konsequenz oder Verlust führt

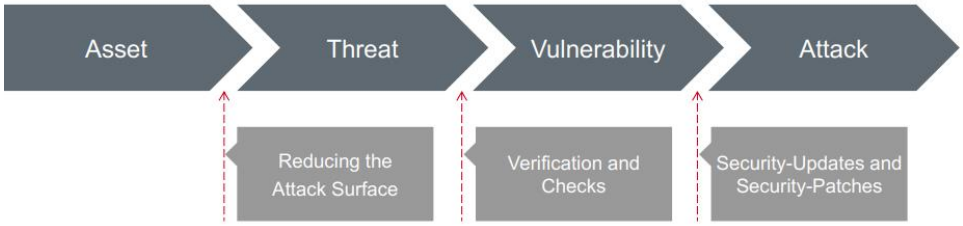
Exploit: Ausnutzung

Vulnerability: Eine Schwachstelle oder Fehler, dessen Exploit zu einem Verlust oder Konsequenz führt

Consequence: Konsequenz oder Verlust

Attack
An Attack is a particular attempt to violate at least one *(or more)* security property of an asset.

The path from an asset to an attack *(via threat and vulnerability)*



Recommendations
- **Integrate security early** at the beginning
- **Consider security** in the **lifecycle**, continuously
- **Understand** and **apply** both **organizational** and **technical security controls**
- **Take** the **operational environment into account**
- **Decide** on a **risk basis** about measures

"Complexity is an enemy of security"

"Profound architecture is the basis of secure software"

"IT security objectives may conflict with another"

STRIDE-Prinzip

- **S**poofing (Identität verschleiern)
- **T**ampering (Manipulation)
- **R**epudiation (Verleugnung)
- **I**nformation disclosure (Verletzung Privatsphäre)
- **D**enial of Service
- **E**levation of privelidge (Rechts ausweiten)

Failure, Fault, Error and Vulnerability

Failure(Ausfall): Abweichung des Produkts von geforderter Funktionalität, extern beobachtbar

Fault(Störung): Fehler des Softwareprodukts, z.B. falscher Prozess, intern beobachtbar (Bug), Fault-Masking (Fehlermaskierung!)

Hinweis: **Fault Masking** tritt auf, wenn ein oder mehrere Faults andere Faults verbergen.

Error(Fehler)**:** Ursache für Faults, Bsp.: vom Programmierer oder Anforderungsingenieur verursacht

Vulnerability(Schwachstelle)**:** bestimmte Errors, die vom Angreifer ausgenutzt werden können

Weakness vs. Vulnerability

Weakness/Schwachstelle:

- systematischer Fehler oder Mangel in einem System

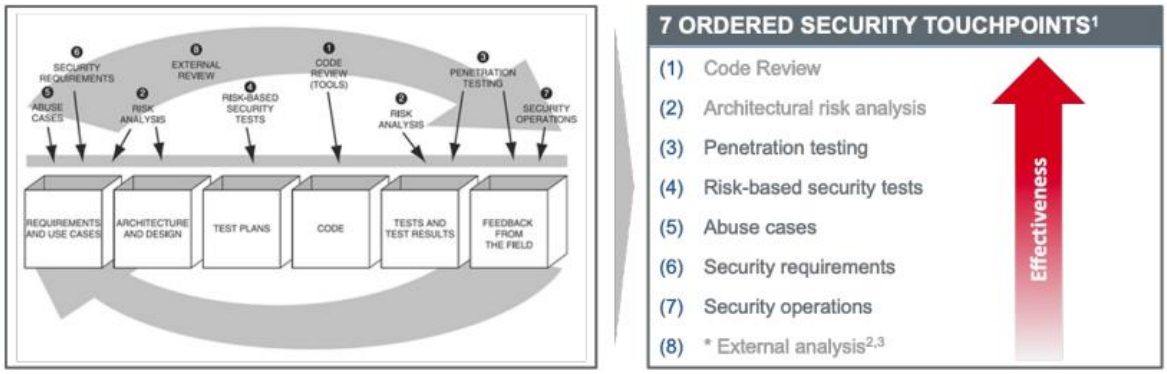
- grundsätzliche Probleme
- Identifikation und Behebung sind entscheidend für die Stärkung der IT-Sicherheit.

Vulerability/Verwundbarkeit:

- konkrete Ausnutzbarkeit dieser Schwachstelle durch Angreifer
- ausnutzbaren Aspekte

Lifecycles

7+1 Digital Security Touchpoints



- Apply security engineering best practices to artifacts and adopt the touchpoints
- Obtain priorities and decide the touchpoint ranking to remove bugs and flaws

OWASP (Open Web Application Security Project)

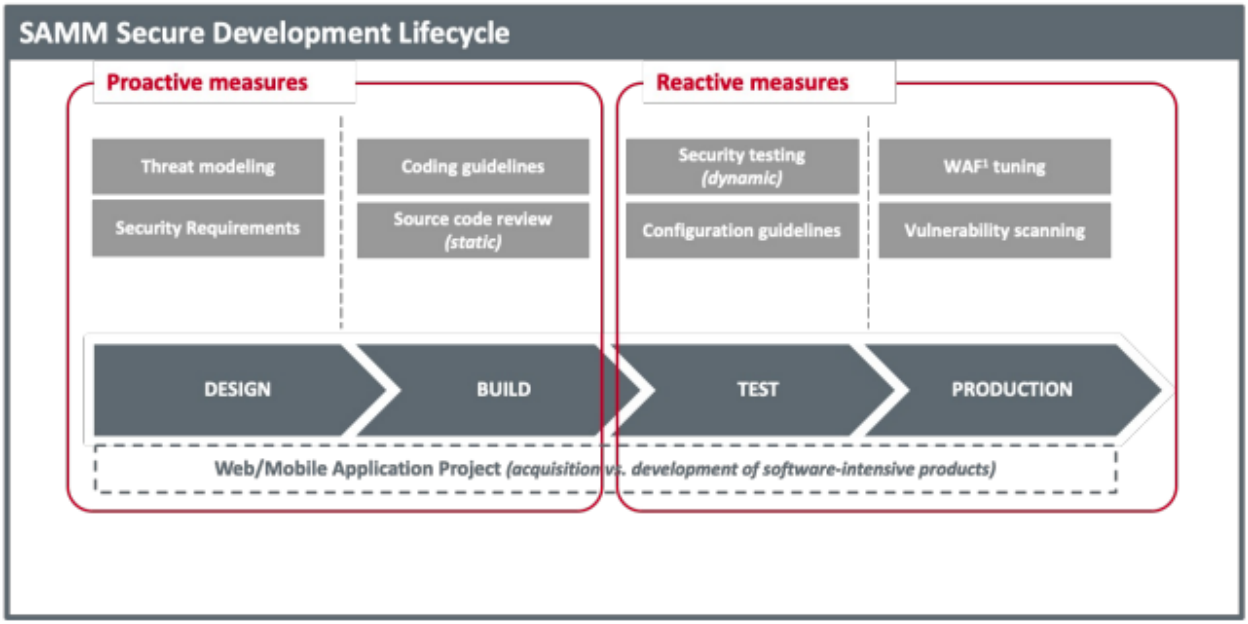
Non-Profit-Organisation mit dem Ziel, die Sicherheit von Anwendungen und Diensten im World Wide Web zu verbessern.

OWASP SAMM (Software Assurance Maturity Model)

OWASM SAMM ist der Nachfolger von OWASP CLASP und ist eine Strategie, um Software vor Organisations-spezifischen Risiken zu schützen. Es ist darauf ausgerichtet, die Zusammenarbeit zwischen den Geschäfts-, Entwicklungs- und Sicherheitsabteilungen zu fördern, um eine ganzheitliche Sicherheitskultur zu etablieren. OWASP SAMM steht für "Software Assurance Maturity Model" und ist ein Rahmenwerk, das entwickelt wurde, um Organisationen dabei zu helfen, ihre Software-Sicherheitspraktiken zu bewerten und zu verbessern. Das Modell konzentriert sich auf drei Hauptkomponenten:

2. Geschäftspraktiken (Business Functions): Hierbei geht es darum, wie Sicherheit in die Geschäftsprozesse integriert ist.
2. Software-Engineering-Praktiken (Software Engineering Practices): Dies bezieht sich auf die sicherheitsrelevanten Aktivitäten im Entwicklungsprozess, von der Planung bis zur Implementierung.

3. Organisation und Governance (Organizational Governance): Dieser Aspekt betrachtet die Struktur und den Rahmen, den eine Organisation für ihre Sicherheitsbemühungen geschaffen hat.



Design, Architektur und Security Patterns

--> Warum, Welche Probleme das löst, usw.

Security Patterns

Lösen generelle, bekannte Sicherheitsprobleme. Behandelt die Architekturaspekte.

Security Patterns sind Lösungen zu Problemen, die ein Set an Bedrohungen durch das Implementieren von Sicherheitskontrollen adressieren. Security Patterns mindern die Gefahr einer Bedrohung, aber lösen **nicht** die Vulnerability. **Security patterns are directly related to threats (i.e., not to vulnerabilities)**

z.B. Circle of Trust, Identity Provider, Identity Federation, Liberty Alliance Identity Federation

Architektur Pattern

Beschreiben Softwarearchitektur Best Practises

Design Pattern

Codeorientiert, da Sicherheit als Subsystem betrachtet wird

Security goals, properties, objectives

CIA-Prinzip

Confidentiality (Vertraulichkeit): Sichern von Daten durch Verschlüsselung und Zugriffskontrolle

Integrity (Integrität): Zugriffskontrolle, Versionskontrolle, Digitale Signaturen, Hashing, Cycle Redundancy Check --> Erreichen von Akkuratheit, Konsistenz und Vertraulichkeit der Daten

Availability (Verfügbarkeit): Clustering, Load Balance, DDoS-Schutz, RAID & Backup, Redundanz bei Daten, Desaster-/Recovery Plan --> Erreichen von Zuverlässigkeit und dauerhaften Zugang zu Daten durch vertrauliche Parteien

Tradeoffs CIA-Triat:

- Confidentiality vs. Integrity: Stehen eng zusammen. Sind jedoch die Daten nur verschlüsselt, können sie trotzdem von unautorisierten manipuliert, beschädigt werden
- Authenticity vs. Availability: Verwendung von aufwändigen Authentifizierungsverfahren schränkt die Verfügbarkeit bezüglich der Performance ein
- Availability vs. Confidentiality: geht Schlüssel verloren, sind Daten nichtmehr verfügbar
- Authentity vs. Integrity: Angreifer könnten User imitieren und die Vertraulichkeit verletzen

CI5A

Confidentiality, Integrity, Availability, Authentification, Authorization, Accounting and Anonymity

Vertraulichkeit, Integrität/Unversehrtheit, Verfügbarkeit, Authentifizierung, Autorisierung, Buchführung, Anonymität

Security Target, Protection Profile, Target of Evaluation, TSF

Common Criteria:

Standard zur Bewertung der Sicherheit in der Informationstechnologie/ in Softwareprodukten.

Ermöglich unverzerrte Prüfung von Produkten anhand Industrie STANDARDS.

Zum Beschreiben und Evaluieren von Sicherheitsstandards.

Security Functional Requirements (SFR)

Funktionen in den TOE (Target of Evaluation) umsetzen, um Sicherheitsanforderungen zu realisieren

Security Assurance Requirements (SAR)

Maßnahmen, um die Einhaltung der Sicherheit zu gewährleisten (z.B. Updates, …)

Protection Profile: (PP)

- beschreibt das „Was?“
- Implementierungsunabhängige Anforderungen an eine Produktfamilie
- befasst sich mit Bedrohungen, Umweltproblemen, Annahmen, Sicherheitszielen, Sicherheitsanforderungen, wird von Common Criteria ermittelt

Security Target: (ST)

von z.B. Firmen, implementierungsabhängige Antwort auf Protection Profile mit einem speziellen Produkt (Target of Evaluation (ToF))

Target of Evaluation Security Functions

TSF’s werden nicht als Asset sondern als Sicherheitsmechanismen betrachtet => müssen impl. werden!

ISMS

"Informationssicherheitsmanagementsystem" (englisch: Information Security Management System)

- systematischer Ansatz zur Verwaltung und Sicherung von Informationen in einer Organisation
- Die ISO/IEC 2700x-Serie sind internationale Standards, die die Anforderungen für ein ISMS festlegen.
- grundlegende Prinzipien umfassen die Identifikation von Risiken, die Implementierung von Sicherheitskontrollen, die kontinuierliche Überwachung und Verbesserung des Systems
- Diese Standards sollen CIA-Prinzipien für die Datenhaltung von Organisationen sicherstellen

Plan Do Check Act **(PDCA)-Cycle:** Für kontinuierliche Verbesserung der IT-Sicherheit.

https://youtu.be/yGxTuoPt3lE?si=fPUkBUJbmZgTeCBe

Elemente eines ISMS:

ISO 27001

internationaler Standard für Informationssicherheitsmanagement

Merkmale:

Zweck: Festlegung von Anforderungen für ein Informationssicherheitsmanagementsystem (ISMS) zur Gewährleistung von Vertraulichkeit, Integrität und Verfügbarkeit von Informationen.

Anwendbarkeit: Branchenübergreifend für Organisationen jeder Größe.

Ansatz: Betont prozessorientierte Sicherheitsverwaltung mit einem Plan-Do-Check-Act (PDCA)-Zyklus.

Risikobewertung: Fokussiert auf die Identifikation von Bedrohungen und Schwachstellen sowie die Umsetzung angemessener Sicherheitsmaßnahmen.

Kontinuierliche Verbesserung: Betont die Notwendigkeit einer ständigen Überwachung, Überprüfung und Anpassung von Sicherheitsmaßnahmen.

Compliance: Verpflichtung zur Einhaltung gesetzlicher Anforderungen und anderer Vorschriften.

Zertifizierung: Die Zertifizierung nach ISO 27001 zeigt, dass eine Organisation international anerkannte Standards für Informationssicherheit erfüllt.

Arten von Angriffen

Logisch	Physikalisch
Bsp: Malware-Infektionen, Denial-of-Service-Angriffe, Datenmanipulation, unbefugter Zugriff auf Computersysteme, Phishing-Angriffe	Bsp: Eindringen in einen Serverraum, Diebstahl von Hardware, Sabotage physischer Geräte oder Infrastruktur
Finden auf Software-, Daten- oder Netzwerkebene statt	Finden auf Hardwareebene ab
Finden über Netzwerkschnittstellen statt	physische Zugriffe zu erlangen

- Angriffe erfolgen oft in Kombination

Consequence vs Loss

Loss ist der unwiderrufliche Verlust der Daten.

Consequence beschreibt eine Konsequenz, welche aus einem Mangel der Sicherheit in der Datensicherung hervorgeht. Das kann zum Beispiel eine Manipulation oder Beschädigung der Daten sein.

Security Controls

Coding Prinzipien

Authentifizierung: Passwort-Authentifizierung, Logout für ungültige Authentifizierungen

Autorisierung: Zugriffsberechtigungen für Funktionen, Seiten, … prüfen, Zugriffsmechanismen auf Server (und Client) validieren

Verschlüsselung: Keine eigene Verschlüsselung implementieren, etablierte Bibliotheken und Hardware verwenden, Verschlüsseln von Authorisierung und Payloaddaten, Daten mit zufälligem Salt and Seed hashen

Error Handling: implementiere Exception handling, keine geheimen Daten loggen, Bereinigen von Fehlerzuständen

Session Management: geschützte Kommunikationswege erzwingen, Lebensdauer der Sitzung überwachen, Sitzungsmanagement evtl. einführen (z.B. TCP), Ändern der Session-ID nach erfolgreicher Anmeldung erzwingen

Logging: Anmeldedaten nicht loggen, nur das notwendige Loggen, auf hohe Datenqualität beim Loggen achten

Datenvalidierung: Länge von Daten prüfen, vor Verwendung validieren, vor Ausgabe überprüfen

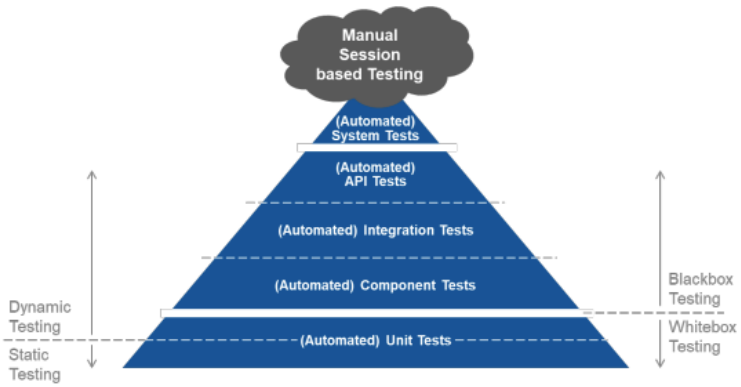
Testpyramide

Je früher Errors gefunden werden, desto besser.

Detect errors early in the SDL and conduct testing quickly *(e.g., parallel)*

Basic test types in terms of risk based security testing

(1) Functional testing
(2) Non-functional testing
(3) Structural testing
(4) Change-related testing
(5) Security testing
(6) Environment testing



[1] Data Source: (Cohn, 2009)

White-Box Testing: Testet alle Implementierungen des Systems, betont Pfadabdeckung, findet Implementierungsfehler, keine Garantie für Spezifikationserfüllung.

Black-Box Testing: Testet Spezifikationen, fokussiert auf Ein- und Ausgaben ohne Zugriff auf interne Strukturen, erkennt Spezifikationsmängel, keine Garantie für korrekte Implementierung.

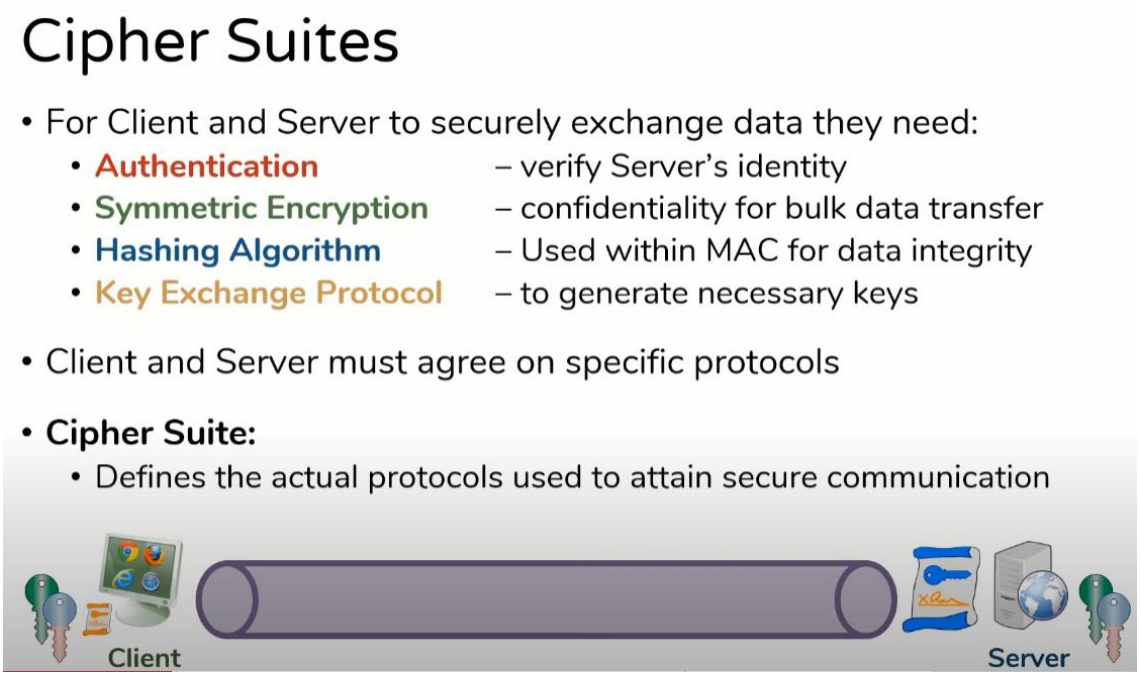
Gray-Box Testing: Kombination von Black- und White-Box Testing, leitet Varianten aus internen Daten ab und führt Tests mit Datenvarianten durch, bietet ausgewogene Herangehensweise für Fehlererkennung.

Pentesting

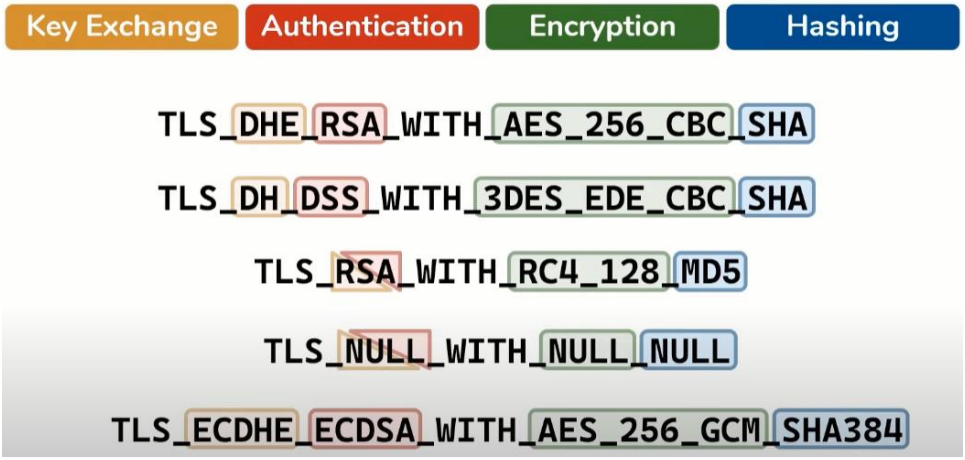
- Im Integration-Test-Layer
- prüft die Sicherheit von Systembestandteilen und Anwendungen
- können Sicherheitslücken aufdecken, aber nicht ausschließen
- nutzt Mittel und Methoden, die notwendig sind, um unautorisiert in das System einzudringen (Penetration)

TLS Ciphersuits

- TLS ist der Nachfolger von SSL (aktuell 1.3 seit 08.2018, RFC 8446)
    1. Meistens noch TLS 1.2
- Wird für Authentifizierung über mehrere Netzwerkknoten verwendet (zB. TCP (unten) und http(s))
- TLS sorgt für **Vertraulichkeit** (symmetrische Verschlüsselung) und **Nachrichtenintegrität** (message authentification code = MAC)
- https = http + SSL/TLS => sichere Webkommunikation



Cipher Suits beschreiben eine konkrete Implementierung der Kommunikation zwischen Server und Client (z.B. **TLS_AES_256_GCM_SHA384**). Also Authentication, Symmetrische Verschlüsselung, Hashing Algorithmus und Schlüssel Austausch Protokoll.



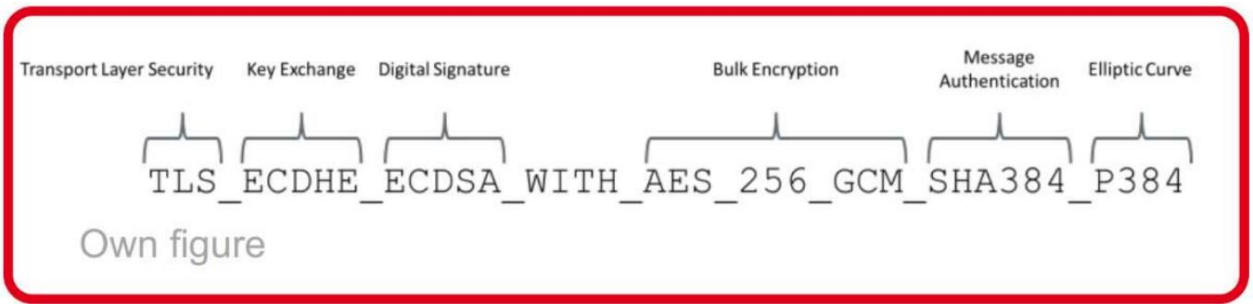
TLS 1.3 Verbesserungen:

- RSA-Schlüsselaustausch und RSA Schlüssel Verschlüsselung entfernt
- Diffie-Hellman als einziger Schlüsselaustauschmechanismus
- Perfect forward secrety (PFS – nicht mehr möglich Langzeitschlüssel vom Sitzungsschlüssel abzuleiten)
- Bevorzugte AEAD (Authentification Encryption with Associated Data) cipher suites (CIA)
- BEAST, CRIME eliminiert
- Schnellerer Handshake

Bsp:

TLS_AES_256_GCM_SHA384:

Diese Cypher suite, die in TLS 1.3 eingeführt wurde, verwendet den AES-256-GCM-Verschlüsselungsalgorithmus und die SHA-384-Hashfunktion. Es erfordert keinen Schlüsselaustauschalgorithmus, da es auf dem während des Handshakes festgelegten Schlüssel basiert



Common weakness enumeration (CWE)

CWE describes a weakness, but not vulnerabilities!

Schwäche, welche ausgenutzt werden kann

Top 25:

- Out-of-bounds Write
- Cross site scripting
- Sql injection
- OS command injection
- Use after free
- Inproper Input Validation
- Out of bounds Read

OWASP SAMM

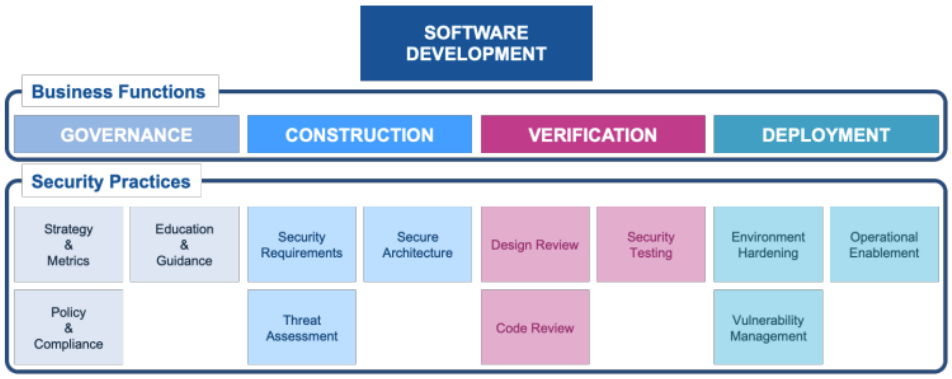
SAMM can be utilized by small, medium and large organizations using any style of (software) development

3 SAMM-Principles[1]

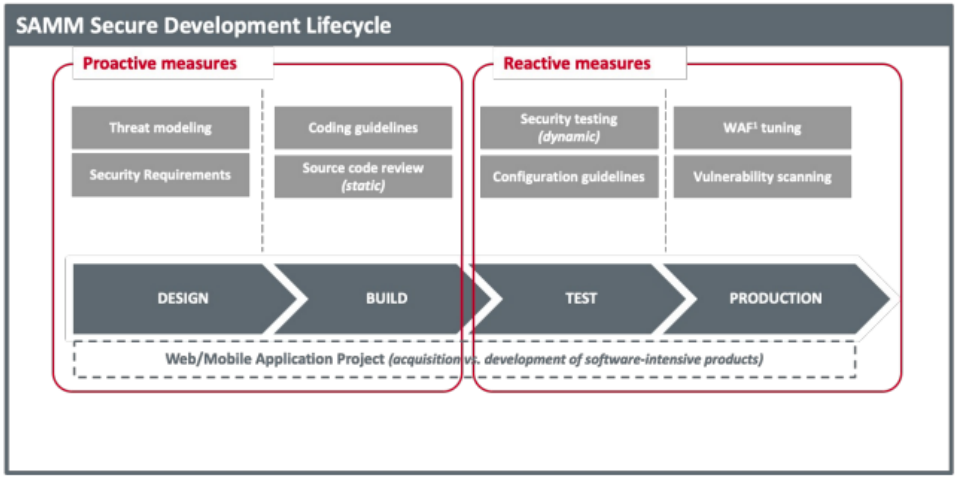
- An organization's behavior changes slowly over time
  A successful software security program should be specified in small iterations that deliver tangible assurance gains while incrementally working toward long-term goals.
- There is no single recipe that works for all organizations
  A software security framework must be flexible and allow organizations to tailor their choices based on their risk tolerance and the way in which they build and use software.
- Guidance related to security activities must be prescriptive
  All the steps in building and assessing an assurance program should be simple, well-defined, and measurable. This model also provides roadmap templates for common types of organizations.

A brief Summary of the SAMM-Structure[1]

- 4 critical Business Functions at the highest level
  Each business function is a category of activities related to software development
- 3 Security Practices for each Business Function
  Each security practice is an area of security-related activities to build assurance for the associated business function as an improvement measure
- 3 Maturity Levels[2] for each Security Practice as Objectives
  Each level is characterized by objectives defined by specific activities and success metrics[3]



Governance	Construction	Verification	Deployment
Strategy & Metrics Overall strategic direction of the software assurance program & instrumentation of processes & activities to collect metrics about an organization's security posture	**Threat Assessment** Identify and characterize potential attacks on software to better understand the risks and facilitate risk management	**Design Review** Inspect artifacts created from the design process to ensure provision of adequate security mechanisms and adherence to expectations for security	**Vulnerability Management** Establish consistent process for managing internal and external vulnerability reports to limit exposure and gather data to enhance the security assurance program
Policy & Compliance Set up a security and compliance control framework and audit framework to achieve increased assurance in software under construction and in operation	**Security Requirements** Promote the inclusion of security-related requirements during the software development process to specify correct functionality from inception	**Code Review** Assess source code to aid vulnerability discovery and related mitigation activities as well as establish a baseline for secure coding expectations	**Environment Hardening** Implement controls for the operating environment in which software executes to bolster the security posture of applications that have been deployed
Education & Guidance Increase security knowledge among personnel in software developement through training and guidance on security topics relevant to individual job functions	**Secure Architecture** Bolster the design process with activities to promote secure-by-default designs and control over technologies and frameworks upon which software is built	**Security Testing** Test software in ist runtime environment in order to discover vulnerabilities and establish a minimum standard for software releases	**Operational Enablement** Identify and capture security-relevant information needed by an operator to properly configure, deploy, and run software



Relevant Attack Models

Ciphertext only attack

- The attacker only knows the ciphertext
- The attacker intercepts the ciphertext on the channel

Known plaintext attack

- The attacker knows the plaintext *(i.e., the secret message)*
- The attacker knows the ciphertext *(i.e., the encrypted form of the secret message)*

Chosen plaintext attack

- The attacker e.g., owns the cryptosystem
- The attacker can choose the plaintext
- The attacker is able to derive the ciphertext

and others *(e.g., exhaustive key search, related key)*

Computational security *(or hardness)*

- Resistant against computationally bounded attackers
- Resistant against bounded attackers in terms of knowledge

Provable security *(or hardness)*

- Resistant against computationally bounded attackers
- There exist mathematical proofs

Unconditional security *(or hardness) (i.e. information theoretic)*

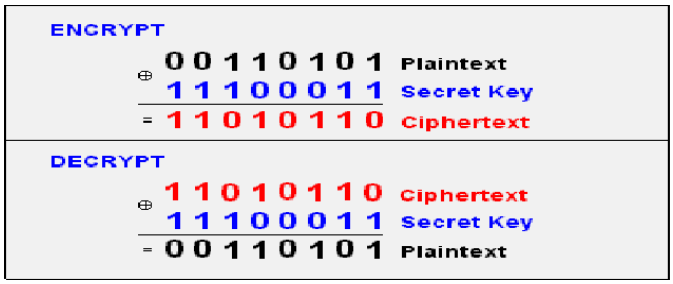
- Resistant against an arbitrary instance of an attacker
- The ciphertext provides no information about the plaintext

e.g., One-Time-Pad

Logical Operations are Performed on Data

One-time pad

Das Einmalschlüssel-Verfahren (One-Time Pad) wird geheim gehalten und nur einmal verwendet, um eine Nachricht mit einem zufälligen Schlüssel derselben Länge zu schützen. Dieses Verfahren basiert darauf, dass der Schlüssel jedes Mal, wenn er verwendet wird, einzigartig ist und nach der Verschlüsselung sofort verworfen wird. Dadurch wird die Sicherheit der verschlüsselten Nachricht maximiert, vorausgesetzt, dass der Schlüssel geheim gehalten und nicht wiederverwendet wird.



Grundlagen zu One-time-passwords

NICHT ZU VERWECHSELN MIT ONE-TIME-PADs!

Oft bei Multifaktor-Umgebungen verwendet. Löst das SSO-Problem, dass immer gleiche Passwörter für verschiedene Anwendungen. OTP-Generation durch Synchronisation von Client und Server.

Encryption vs Signature

Signaturen bedeuten nicht verschlüsselung. Verschlüsselung verwendet Schlüsselpaare. → Dient zur Confidentiality/Vertraulichkeit.

Signatur verwendet einmalige Signatur, welche verifiziert werden muss. ➔ Dient nur zur Integrität und Authentifizierung.

ENCRYPTION VS. SIGNATURE
WHAT IS THE DIFFERENCE?

Encryption

- Confidentiality
- Keypair *(sender and receiver)*
- K_e not necessarily the same as K_d

General Process

- Alice computes K_e
- Alice transmits $K_e(M)$
- Bob decrypts $K_d(K_e(M))$

Signature

- Authenticity
- Integrity
- Non-repudiation
- Keypair *(KV,KS)*

General Process

- Signatory signs $K_S(M)$
- Verifier obtains K_V
- Verifier checks $K_V(K_S(M))$

Hash Funktionen

BUT WHAT IS A HASH-FUNCTION?
EXPLANATION

Hash functions *(h)* are one-way functions such that

- $y=h(x)$ *(computationally easy to derive)*
- $x=h^{-1}(y)$ *(computationally hard to derive i.e., computationally impossible)*
- $h(x) \neq h(x')$ *(collision resistence; hard to find (x,x') with x!=x' s.t.* $h(x)=h(x')$*)*

Not all hash functions are collision resistant !

A hash *(value)* is a *(digital)* fingerprint of a document

2 documents with different content should not have the same fingerprint

Secure and safe software

In general: The Developer/Company is responsible for providing secure and safe software with minimum or no known vulnerabilities

Safe Programming: Software should not cause **damage to the environment**

- Privilege escalation
- Attack something else
- Include Backdoors

Secure Programming: Software should be protected from all kinds of deficits[1],resistant against certain attacks or something that can cause harm to the software itself or to a system using it.

Secure coding principles

Security Control	The way to go by
Authentication	▪ Implement password security and account lockout for unsuccessful authentication attempts ▪ Considser unsecure „I forgot my password“ mechanisms ▪ Enforce POST methods for web applications
Authorization	▪ Verify authorization to gain access (for functions, pages etc.) ▪ Check the access context ▪ Validate security mechanisms on the server (client and server)
Cryptography	▪ Do not write your own crypto and use established libraries (and hardware as well) instead ▪ Encrypt both authentication data and payload data assets ▪ Hash data (e.g., passphrases) with salt and seed random number generators, appropriately
Error Handling	▪ Detect and process deficits and implement exception handling ▪ Keep secret data private (also in logfiles) and do not disclose data, unnecessarily ▪ Cleanup error states and error conditions
Session Management	▪ Ensure protected communication channels and enforce a reasonable session lifespan ▪ Decide when session management is necessary and leverage existing solutions (e.g., TCP) ▪ Force a change of sessionID after a successful login (client and server)
Logging	▪ Avoid logging sensitive data (e.g., credentials, passphrases) and prepare for log spoofing ▪ Minimize the amount of data to be logged to the absolute necessity ▪ Increase the quality of logged data and reduce data quality issues
Data Validation	▪ Endorse the length of data before writing it to buffers ▪ Validate data before using (insert, output, take, accept) ▪ Verify data before sending/broadcasting to clients

Open Web Application Security Project macht jedes Jahr OWASP Top 10 vulnerabilities und co. Daran orientiert sich Cyber Security Branche. Außerdem gibt’s da Guidelines für Entwickler für Secure Coding usw.