Computersystemsicherheit

Zusammenfassung

Tim Pollandt, mit Korrekturen von Janika Krull

9. März 2022

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Allgemeines

1.1 Arten von Sicherheit

Grundlegend wird zwischen Betriebssicherheit (engl. *safety*) und Angriffssicherheit (engl. *security*) unterschieden.
Angriffssicherheit ist der Schutz vor aktiven Angreifern mit Schadabsicht.
Betriebssyicherheit ist der Schutz vor Fehlern innerhalb des Systems, das Sichern der Zuverlässigkeit und der Schutz vor äußeren Faktoren wie Umwelteinflüssen.

1.2 Schutzziele der Angriffssicherheit

Man versucht die folgenden Schutzziele zu erreichen:

- **Authentizität**: Die empfangenen Daten stammen von dem Absender, mit dem man kommunizieren möchte.
- **Integrität**: Die Daten wurden auf dem Weg nicht verändert und stammen genau so vom Absender.
- **Vertraulichkeit**: Die Daten können auf dem Kommunikationsweg nicht von anderen gelesen werden.
- **Nicht-Abstreitbarkeit**: Dem Absender kann nachgewiesen werden, dass er die Daten verschickt hat.
- **Verfügbarkeit**: Das System soll jederzeit verfügbar sein und vor Ausfall geschützt sein.

1.3 Schutzziele der Betriebssicherheit

Man versucht die folgenden Schutzziele zu erreichen:

- **Schutz vor Ausfall**: bespielsweise soll ein Signal bei Stromausfall nach unten klappen und damit „Halt“ (Hp 0) zeigen, um größere Schäden zu verhindern.
- **Redundanz**: Kritische Systeme sollten bei Ausfall durch Backupsysteme ersetzt werden können. Daten sollten gesichert sein.
- **Schutz vor Architektur-/Softwarefehlern**: beispielsweise sollten Overflows von Variablen verhindert werden.

1.4 Angriffe

Die meisten Angriffe haben finanzielle Interessen. Nutzerdaten und Gelder lassen sich digital übernehmen und sind wertvoll.
Mögliche Angriffe sind:

- **Phishing**: Ausliefern einer ähnlich aussehenden, gefälschten Website, damit Nutzer ihre Zugangsdaten eingeben.
- **Man-in-the-Middle**: Abgreifen von Daten, indem man sich in die Mitte der Verschlüsselungsverbindung stellt und mit jedem Partner einzeln verschlüsselt (und vortäuscht, der jeweils andere zu sein).

2 Kryptographie

Durch Kryptographie können die oben genannten Schutzziele erreicht werden.
Außerdem ist die Kryptoanalyse, also das Nachweisen der Sicherheit und Angreifen auf Kryptographieverfahren, in diesem Fall Teil der Kryptographie.

2.1 Begriffe

- **Klartext**: die unverschlüsselte Nachricht
- **Klartextraum**: die Menge aller möglichen Klartexte
- **Chiffrat**: die verschlüsselte Nachricht
- **Chiffratraum**: die Menge aller möglichen verschlüsselten Nachrichten
- **Verschlüsselung**: der Vorgang der Umwandlung eines Klartexts in ein Chiffrat
- **Entschlüsselung**: der Vorgang der Umwandlung eines Chiffrats in einen Klartext
- **Schlüssel**: ein Geheimnis, das benötigt wird, um eine Nachricht ver-/entschlüsseln zu können
- **Schlüsselraum**: die Menge aller möglichen Schlüssel
- **Kryptosystem**: eine Ver-/Entschlüsselungsvorschrift mit zugehörigem Klartextraum, Chiffratraum und Schlüsselraum

2.2 Mathematische Grundlagen

Die hier nicht weiter beschriebenen mathematischen Grundlagen sind wie in der Mathematik üblich definiert.

2.2.1 Euklidischer Algortithmus

Einfacher euklidischer Algorithmus:

Seien $a \neq 0, b \in \mathbb{Z}$.
Dann liefert der folgende Algorithmus den größten gemeinsamen Teiler von a und b:

```
a = |a|; b = |b|;
while (b > 0):
    if (a > b):
        a = a - b;  | Alternativ: a = a mod b;
    else:
        b = b - a;  | Alternativ: b = b mod a;
return a;
```

Erweiterter euklidischer Algorithmus:

Seien $a, b \neq 0 \in \mathbb{Z}$.
Dann liefert der folgende Algorithmus den größten gemeinsamen Teiler von a und b:

```
a_0 = max(|a|, |b|); b_0 = min(|a|, |b|); n = 0;
while (b_n > 0):
    q_n = (int) a_n / b_n;
    r_n = a_n / b_n - q_n; // a_n mod b_n
    a_{n+1} = b_n; b_{n+1} = r_n;
    n = n + 1;
return a;
```

Nun gilt:
$ggT(a,b) = a_n = b_{n-1} = r_{n-2} = a_{n-2} - q_{n-2}b_{n-2} = b_{n-3} - q_{n-2}r_{n-3} = ... = x \cdot a_0 + y \cdot b_0 = ax + by$

Der EEK lässt sich auch in Tabellenform lösen:
(a, b, q intuitiv nach unten; dann von unten $s = t_{alt}$, $t = s_{alt} - q \cdot t_{alt}$)

a	b	q	s	t
99	78	1	-11	14
78	21	3	3	-11
21	15	1	-2	3
15	6	2	1	-2
6	3	2	0	1
3	0		1	0

a	b	q	s	t
61	13	4	3	-14
13	9	1	-2	3
9	4	2	1	-2
4	1	4	0	1
1	0		1	0

$99 \cdot (-11) + 78 \cdot 14 = -1089 + 1092 = 3$
$61 \cdot 3 + 13 \cdot (-14) = 183 - 182 = 1$

2.2.2 Lemma von Bézout

$\forall a, b \in \mathbb{Z} : \exists x, y \in \mathbb{Z} : ax + by = ggT(a,b)$

2.2.3 Eulersche Phi-Funktion

$\varphi(b) := |\{a \in \mathbb{N} | a < b, ggT(a,b) = 1\}|$

Es gilt (p prim):
$\varphi(p) = p - 1$
$\varphi(p^n) = p^n - p^{n-1}$
$\varphi(ab) = \varphi(a) \cdot \varphi(b)$

Euler-Fermat:
$\forall m, n \in \mathbb{N}$: **m,n teilerfremd** $\Rightarrow m^{\varphi(n)} mod n = 1$

2.3 Kerckhoffs Prinzipien

Diese Prinzipien sollte jedes Kryptosystem erfüllen.

1. Das System ist zumindest praktisch unknackbar. Ohne Kenntnis des Schlüssels ist der Klartext nicht innerhalb eines praktikablen Zeitraums errechenbar. Optimalerweise ist die Sicherheit mathematisch beweisbar.
2. Die Geheimhaltung des Verfahrens ist für die Sicherheit nicht erforderlich. Das gemeinsame Kennen eines Geheimnisses (Schlüssel) reicht aus, um Sicherheit zu garantieren.
   In der Vergangenheit hat sich gezeigt, dass öffentliche Verfahren, bei denen jeder die Sicherheit prüfen kann, zu mehr Sicherheit führen als geheimgehaltene Verfahren.
3. Der Schlüssel muss ohne Hilfe geschriebener Notizen kommunizierbar und aufbewahrbar sein und muss nach Belieben der Kommunikationspartner ausgewechselt oder modifiziert werden können.
   Dieses Ziel ist heute nicht mehr umsetzbar und wird weitgehend nicht mehr beachtet.
4. Es muss auf telegraphische Kommunikation anwendbar sein.
5. Es soll portabel sein und nicht voraussetzen, dass sich mehrere Menschen treffen.
6. Es muss einfach von jedem anwendbar sein.

2.4 Angriffe

Diesen Angriffsszenarien sollte eine sichere Verschlüsselung standhalten können:

1. **known chiffretext**: Der Angreifende kennt den verschlüsselten Text (Chiffrat)
2. **known plaintext**: Der Angreifende kennt den verschlüsselten Text (Chiffrat) und den Klartext einer alten Nachricht und versucht nun mithilfe des Chiffrats einer neuen Nachricht, diese zu entschlüsseln.
3. **chosen plaintext**: Der Angreifer kann Chiffrate zu beliebigen Klartextnachrichten mithilfe des Verfahrens generieren.
4. **chosen chiffretext**: Der Angreifer kann Klartexte zu beliebigen Chiffraten mithilfe des Verfahrens generieren.

2.5 Symmetrische Kryptosysteme

Symmetrische Verschlüsselungsverfahren sind daran erkennbar, dass es eine Verschlüsselungsfunktion $e : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ und eine Entschlüsselungsfunktion $d : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$ gibt.

2.5.1 Verschiebechiffre (Cäsar, ROT13)

Sei z die Menge der im Klartext zugelassenen Zeichen und $|z| = n \in \mathbb{N}$. Nun lege man eine Bijektion f zwischen den Zeichen und $[0; n-1] \in \mathbb{N}$ fest. Dann berechne man für den Klartext z^x für jedes einzelne z das verschlüsselte Zeichen $e := f^{-1}(f(z) + s \pmod n)$ für einen gegebenen Schlüssel $s \in [0; n-1] \in \mathbb{N}$. Dann ist $z = f^{-1}(f(e) - s \textbf{ (mod } n\textbf{)})$.
Das Verfahren ist jedoch nicht sicher, da die Schlüsselmenge zu gering ist, Häufigkeitsanalyse möglich ist und merere weitere Angriffsverfahren (s. oben) funktionieren.

2.5.2 Vignière-Chiffre

Die Vignière-Chiffre ist im Prinzip eine Verschiebechiffre mit verschiedenem Schlüssel für jeden Buchstaben. Damit der Schlüssel kürzer als die Nachricht ist, wiederholt man diesen jedoch alle paar Zeichen, wodurch eine Häufigkeitsanalyse zwar schwerer aber noch möglich ist.

2.5.3 One-Time-Pad

Bei einem One-Time-Pad hat der Schlüssel die gleiche Länge wie die Nachricht (jeweils in Binärdarstellung). Nun berechnet man die n-te Stelle der verschlüsselten Nachricht meist durch Anwenden von XOR auf die n-te Stelle der Eingabe und der n-ten Stelle des Schlüssels. Dieses Verfahren ist kryptographisch sicher. Der Klartext lässt sich auf die gleiche Weise mit dem Schlüssel wieder berechnen.

2.5.4 Blockchiffre

ECB: Electronic Codebook

Der Klartext wird in Blöcke fester Länge zerlegt und auf jeden einzelnen Block wird eine Verschlüsselungsfunktion (z. B. ein One-Time-Pad) mit gleichem Schlüssel angewendet. Nötigenfalls wird die Klartextnachricht vorher mit Binärwerten aufgefüllt. Dieses Verfahren ist offensichtlich nicht sicher (s. verschlüsseltes TU Darmstadt-Logo in den Vorlesungsfolien).

CBC: Cipher Block Chaining

Vor der Verschlüsselung des ersten Blocks wird hier noch ein (nicht geheimer) Schlüssel aus Zufalsswerten (sog. Initialisierungsvektor) addiert (XOR) und anschließend dieser Teil verschlüsselt. Der verschlüsselte Teil dient nun als IV für den nächsten Block usw.
Wenn jeder Schlüssel nur bei einer Verschlüsselung eingesetzt wird, ist dieses Verfahren sicher.
Zur Entschlüsselung wird jeder Ciphertext erst entschlüsselt und anschließend der Ciphertext des vorhergehenden Blocks addiert (XOR) (beim ersten Block wird der IV verwendet). Dadurch ist die Entschlüsselung im Gegensatz zur Verschlüsselung parallelisierbar.

CTR/CTM: Counter Mode

Im CTR Mode wird erst die Verschlüsselung auf das Resultat einer Operation auf einer Nonce und einem mit jedem Block inkrementierten Counter angewendet. Anschließend wird diese auf den Klartext addiert (XOR). Die Entschlüsselung funktioniert entsprechend.
Dieser Modus ist eine Stromchiffre.

2.6 Asymmetrische Kryptosysteme

Bei asymmetrischen Verschlüsselungsverfahren hat jeder einen privaten und einen öffentlichen Schlüssel. Mithilfe des öffentlischen Schlüssels können Nachrichten an die Person verschlüsselt werden. Mithilfe des privaten Schlüssels können sie wieder entschlüsselt werden.

2.6.1 RSA

Schlüsselgenerierung:

1. Wähle Primzahlen p und q.
2. Berechne $n = p \cdot q$ und $\varphi(n) = (p-1) \cdot (q-1)$.
3. Wähle e mit $e < \varphi(n)$ und ggT$(e, \varphi(n)) = 1$.
4. Wähle d mit $d < \varphi(n)$ und $e \cdot d \mod \varphi(n) = 1$.

Ver-/Entschlüsselung:

- Verschlüsselung: $c = m^e \mod n$
- Entschlüsselung: $m = c^d \mod n$

2.6.2 ElGamal

Schlüsselgenerierung:

- Wähle zyklische Gruppe $\mathcal{G} = (G, \circ, e)$ und $g, a \in \{2, ..., ord(G) - 1)$. Berechne nun $A = g^a$.
- Dann ist der private Schlüssel $(\mathcal{G}, g, a)$.
- Dann ist der öffentliche Schlüssel $(\mathcal{G}, g, A)$.

Verschlüsselung:

1. Wähle $r \in \{2, ..., ord(G) - 1)$ und berechne $R = g^r$.
2. Berechne $K = A^r = (g^a)^r$ und $C = m \circ K$.
3. Übermittle C und R.

Entschlüsselung:

1. Berechne $K = R^a = (g^r)^a = (g^a)^r$ und $C = m \circ K$.
2. Berechne K^{-1} in $\mathcal{G}$.
3. Berechne $C \circ K^{-1} = m \circ K \circ K^{-1} = m$.

2.7 Hybride Verschlüsselung

Bei Hybrider Verschlüsselung wird der Schlüssel für die symmetrische Verschlüsselung mit asymmetrischer Verschlüsselung verschlüsselt, wodurch die langsamere, asymmetrische verschlüsselung weniger verschlüsseln muss. Die Sicherheit ist dadurch aber von beiden Kryptosystemen abhängig.

2.8 Hash und Signatur

2.8.1 Definition

Eine **Hashfunktion** ist für $n \in \mathbb{N}$ eine Funktion $h : A^* \to A^n$.
Eine **Kompressionsfunktion** ist für $m, n \in \mathbb{N}, n < m$ eine Funktion $h : A^m \to A^n$.

Es geht jeweils darum Nicht-Abstreitbarkeit, Authentizität und Integrität zu garantieren.

2.8.2 Kollisionsresistenz

Eine Funktion heißt **schwach kollisionsresistent**, wenn es zu gegebenem x_1 schwer ist, ein x_2 zu finden mit $h(x_1) = h(x_2)$.
Eine Funktion heißt **stark kollisionsresistent**, wenn es schwer ist, x_1 und x_2 zu finden mit $h(x_1) = h(x_2)$.

2.8.3 Merkle-Darmgård

Mithilfe dieser Konstruktion lässt sich aus einer Kompressionsfunktion eine Hashfunktion kreieren:
Sei A ein Alphabet und $f : A^{n+m} \to A^n$ eine Kompressionsfunktion.
Sei $pad : A^* \to (A^m)^*$ eine Auffüllfunktion.
Sei $h_0 \in A^n$ ein beliebiger Initialisierungsvektor und $g : A^n \to A^n$ eine Finalisierungsfunktion.
Dann sei $h : A^* \to A^n$ für $x \in A^*$ wie folgt zu berechnen:

1. $x_1 \cdot x_2 \cdot ... \cdot x_k = pad(x)$ mit $x_i \in A^m$ für $i \in [1, k] \subset \mathbb{N}$.
2. $h_i = f(append(h_{i-1}, x_i))$ für $1 \leq i \leq k$.
3. $h(x) = g(h_k)$.

2.8.4 RSA-Signatur

Wendet man auf eine Nachricht m eine Hashfunkion h mit Zielbereich zwischen 0 und n an, so erhält man durch $s = h(m)^d \mod n$ die Signatur. Mit dem öffentlichen Schlüssel kann dann jeder prüfen, ob die Signatur korrekt ist, denn dies ist genau dann der Fall, wenn $h(m) = s^e \mod n$.
Das Hashen ist hier unbedingt erforderlich, da man sonst aus (m, s) eine neue signierte Nachricht (s^e, s) generieren könnte, ohne den privaten Schlüssel zu kennen.

2.8.5 DSA: Digital Signature Algorithm

Parametergenerierung

1. Wähle Primzahl q und sehr große Primzahl p mit $q \mid (p - 1)$.
2. Für ein $1 < h < p - 1$, berechne $g = h^{\frac{p-1}{q}}$. Es ist also $ord(g) = q$.
3. p, q und g können öffentlich sein und von mehrere Nutzern verwendet werden.

Schlüsselgenerierung

1. Wähle x mit $1 < x < q$ zufällig.

2. Berechne $y = g^x \mod p$.
3. y ist der öffentliche Schlüssel, x ist der private Schlüssel.

Signieren

1. Wähle k mit $1 < k < q$.
2. Berechne $r = max((g^k \mod p) \mod q, 1)$.
3. Berechne $s = max(k^{-1} \cdot (H(m) + r \cdot x) \mod q, 1)$.
4. Sie Signatur ist nun das Tupel (r, s).

Verifikation

1. Berechne $w = s^{-1} \mod q$.
2. Berechne $u_1 = H(m) \cdot w \mod q$.
3. Berechne $u_2 = r \cdot w \mod q$.
4. Berechne $v = (g^{u_1} \cdot y^{u_2} \mod p) \mod q$.
5. Akzeptiere wenn $v = r$.

2.9 Schlüsselaustausch/-generierung

2.9.1 Needham-Schroeder (symmetrisch)

Annahme: Alle haben mit T einen Schlüssel ausgetauscht. A und B wollen kommunizieren. N_A und N_B sind Nonce.

1. $A \to T$: A, B, N_A
2. $T \to A$: $\{N_A, K, B, \{K, A\}_{K_B}\}_{K_A}$
3. $A \to B$: $\{K, A\}_{K_B}$
4. $B \to A$: $\{N_B\}_K$
5. $A \to B$: $\{N_B - 1\}_K$

Wenn ein Angreifer einen alten Schlüssel hat, kann er allerdings mit B eine Verbindung aufbauen und sich als A ausgeben. Dies kann durch einen Zeitstempel verhindert werden.

2.9.2 Needham-Schroeder (asymmetrisch)

Es gibt auch eine asymmetrische Variante (K_{P_A} und K_{P_B} sind öffentliche Schlüssel, K_{S_T} ist ein privater Signaturschlüssel):

1. $A \to T$: A, B
2. $T \to A$: $\{K_{P_B}, B\}_{K_{S_T}}$
3. $A \to B$: $\{N_A, A\}_{K_{P_B}}$
4. $B \to T$: B, A
5. $T \to B$: $\{K_{P_A}, A\}_{K_{S_T}}$
6. $B \to A$: $\{N_A, N_B\}_{K_{P_A}}$
7. $A \to B$: $\{N_B\}_{K_{P_B}}$

Wenn ein Angreifer einen alten Schlüssel hat, kann er allerdings mit B eine Verbindung aufbauen und sich als A ausgeben. Dies kann durch einen Zeitstempel verhindert werden.

2.9.3 Diffie-Hellmann

Funktionsweise:

1. Wähle zyklische Gruppe $\mathcal{G}$ der Ordnung p (prim) und einen Generator $g \in \mathcal{G}$.
2. A wählt $0 < a < p$ und berechnet g^a. B wählt $0 < b < p$ und berechnet g^b.
3. g^a und g^b werden ausgetauscht.
4. Beide können nun g^{ab} berechnen.

Allerdings sind Vorberechnungen bei bekannter zyklischer Gruppe für selbige möglich.
Um MitM-Angriffe zu verhindern, sollten die Nachrichten signiert werden (Station-to-Station).

2.10 AE(AD): Authenticated Encryption (with Associated Data)

AE: Daten sollen verschlüsselt und unmanipulierbar sein.
Es gibt verschiedene Möglichkeiten:

1. EtM (Encrypt-then-MAC): [Bsp.: IPSec] Nachricht wird verschlüsselt. Anschließend wird dies und der Hash davon übertragen.
2. MtE (MAC-then-Encrypt): [Bsp.: TLS] Nachricht und deren Hash werden zusammen verschlüsselt.
3. E&M (Encrypt-and-MAC): [Bsp.: SSH] Nachricht wird verschlüsselt und Hash der Nachricht angefügt.

AEAD: Unverschlüsselte Metadaten sollen zusätzlich übertragen werden. Die Integrität soll weiterhin gewährleistet sein.

2.11 Secret Sharing

Daten sollen nur entschlüsselt werden können, wenn t von n Beteiligten mitwirken.
Möglichkeit: $\binom{n}{t}$ verschiedene Verschlüsselungen

2.11.1 Shamirs Secret Sharing Protokoll

Sei k der Schlüssel. Es gebe n Shareholder mit Thresholt t.
Es wird zentral vom *Dealer* eine Primzahl p mit $p > k$ und $p > n$ gewählt. Wir rechnen nun in $\mathbb{Z}_p$.
Der Dealer wählt ein Polynom $\sum_{i=0}^{t-1} f_i \cdot x^i$ mit $f(0) = k$.
Nun verteilt der Dealer $\{s_1, s_2, ..., s_n\} = \{(1, f(1)), (2, f(2)), ..., (n, f(n))\}$ an die einzelnen Beteiligten.

3 IT-Sicherheit und Zuverlässigkeit

3.1 Zugriffsrechte

3.1.1 DAC: Discretionary Access Control

Hier ist der Eigentümer für die Rechtevergabe verantwortlich. Für jede Kombination aus Zugreifendem und Datei können alle Rechte explizit gesetzt werden. Rechteänderungen sind allerdings schwer umzusetzen und nur Schreibzugriff auf das eigene Passwort in der Passwortdatei zu gewähren ist schwer. Statt einer Zugriffsmatrix (Datei $\leftrightarrow$ Prozess) kann auch eine Access Control List (ACL) eingesetzt werden, damit Rechte an einer Datei effizient bestimmbar sind. Bei dynamischen Subjektmengen ist dies allerdings schlecht skalierbar. In Unix-Systemen werden die Rechte read, write und execute für Besitzer, Gruppe und Andere gespeichert. Auch eine subjektbezogene Rechtespeicherung ist denkbar, allerdings weitgehend nicht praktikabel.

3.1.2 RBAC: Role-based Access Control

Es wird eine Zuordnung von Subjekten auf Gruppenzugehörigkeit zu allen existierenden Gruppen gespeichert und eine Rechtezuordnung von jeder Gruppe zu möglichen Rechten.
In dieser Modellierung können Rechte mithilfe einer partiellen Ordnung (antisymmetrisch, reflexiv, transitiv) auf den Gruppen auch vererbt werden.

3.1.3 MAC: Mandatory Access Control

Subjekte und Objekte bekommen jeweils Sicherheitsstufen zugewiesen. Danach wird entschieden, auf welche Dateien wer zugreifen darf. Es existiert eine Zuordnung zwischen den Gruppen der Subjekte und den Sicherheitsstufen der Objekte.

3.2 Authentifizierung

3.2.1 Begriffe

- Identität: Menge von Attributen eines Subjekts
- Authentisierung: Bereitstellung von Dokumenten zur Identitätsprüfung
  $\rightarrow$ durch Wissen (PW oder Challenge-Response mit gemeinsamem Schlüssel)
  $\rightarrow$ durch Merkmal (Universalität, Eindeutigkeit, Beständigkeit, Erfassbarkeit, Akzeptanz, Fälschungssicherheit)
  $\rightarrow$ durch Besitz (Kryptoprozessor mit z. B. RSA)
- Authentifizierung/Authentifikation: Prüfung der Echtheit der Unterlagen
- Autorisierung: Gewährung des Zugriffs

3.2.2 Fehlersicherheit

- **FAR**: false positive
- **FRR**: false negative
- **EER**: Equal Error Rate

3.3 Netzwerksicherheit

7. Application Layer
6. Presentation Layer
5. Session Layer
4. Transport Layer
3. Network Layer
2. Data Link Layer
1. Physical Layer

4. Application Layer | HTTP(S), FTP, ...
3. Transport Layer | TCP, UDP, ...
2. Internet Layer | IP, IPSec, ...
1. Link Layer | Ethernet

3.3.1 IP-Subnetze

IP-Adresse: 192.168.0.101
Netzmaske: 255.255.255.0

Network ID: 192.168.0.0
Lokaler Broadcast: 192.168.0.255
Classless Inter-Domain Routing: 192.168.0.101/24
Gateway: 192.168.0.1

3.3.2 SQL-Injection

SQL-Abfrage z.B.:

```
<? php $sql = "SELECT * FROM members WHERE username = '$username'
and password = '$password' "; ?>
```

Dies führt mit username: `D120' or 'x' = 'x` und password: `test123' or '1' = '1` zur Anmeldung. Bei manchen Datenbank-Systemen ist auch durch das Einfügen von `'; DROP TABLE tablename; #` in ein Statement die Tabelle *tablename* löschbar.
Auch der Expression `admin' AND 1=0 UNION ALL SELECT 'admin', MD5('test123')` kann helfen, dass man sich als Benutzer **admin** mit dem Passwort **test123** anmelden kann.

Schützen kann man sich durch Maskieren von Sonderzeichen und nicht Einräumen bestimmter Berechtigungen.

3.3.3 XSS: Cross Site Scripting

Ein Angreifer kann entweder persistent über Formularfelder (zum Beispiel Gästebuch) versuchen, JavaScript in die Website einzubauen oder dem Nutzer eine E-Mail mit einem manipulierten Link schicken, der den Code reflected auf der Website ausführt. Daraufhin bekommt der Angreifer zum Beispiel auf der Website eingegebene, kritische Daten zugeschickt.
Schützen kann man sich, indem man Zeichen aus auf der Website zugänglichen eingegebenen Daten filtert, z. B. mit `htmlspecialchars()`.

3.4 Malware

3.4.1 Trojaner

Trojaner sind Programme, die neben bekannten Funktionen noch unerwünschte Dinge tun, zum Beispiel als Keylogger agieren.

3.4.2 Virus

Viren sind Programme, die ihren Code in Wirtsprogramme einbinden uns sich dadurch verbreiten.
Viren können ihren Code bei der Verbreitung ändern, damit sie von statischen Virenscannern nicht erkannt werden.

3.4.3 Wurm

Würmer sind Programme, die sich autonom selbst verbreiten.

3.4.4 Ransomware

Ransomware ist Software, die die Daten auf dem System verschlüsselt und Lösegeld fordert.

3.4.5 Erkennung

Schützen kann man sich vor Malware durch Detektion statischer Virensignaturen, Sandboxing, dynamische Analyse und Heuristiken.

3.5 Verlässlichkeit von Systemen

Arten von Fehlern:

- **Fault:** Ungewöhnliche Eingabe oder Bedingung, die zu Fehlern im System führen kann.
- **Error:** Abweichung des berechneten Ergebnisses vom Erwartungswert.
- **Failure:** Ausfall eines Systems.

Es gilt:

- $availability = \frac{uptime}{uptime+downtime}$

- MTTF: Mean Time To Failure
- MTTR: Mean Time To Recovery
- $availability = \frac{MTTF}{MTTF+MTTR}$

3.5.1 Verlässlichkeit

Sei T die Zeit, die das System korrekt funktioniert. Dann ist $F(t) = P(T \leq t)$ die Ausfallwahrscheinlichkeit und $R(t) = P(T > t)$ die Zuverlässigkeitsfunktion.

Exponentielles Modell

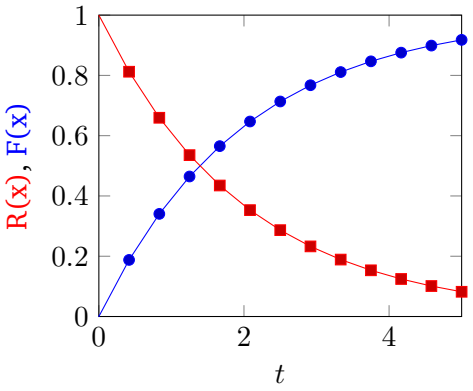
Funktion hat kein Gedächtnis. Systemausfälle sind davon unabhängig, wie lange das System schon läuft, also $P(T > s + t|T > s) = P(T > t)$.

Sei λ die durchschnittliche Anzahl an Fehlern pro Zeiteinheit. Dann ist:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{für } x \geq 0 \\ 0 & \text{für } x < 0 \end{cases}$$

$$F(t) = \int_{-\infty}^{t} f(x)dx = \begin{cases} 1 - e^{-\lambda t} & \text{für } t \geq 0 \\ 0 & \text{für } t < 0 \end{cases}$$

$$R(t) = 1 - F(t) = \begin{cases} e^{-\lambda t} & \text{für } t \geq 0 \\ 1 & \text{für } t < 0 \end{cases}$$



$$MTTF = \int_0^\infty (1 - F(t))dt = \int_0^\infty (e^{-\lambda t})dt = \frac{1}{\lambda}$$

Reihenschaltung:

$$R_{ges}(t) = \prod_{i=1}^{n} R_i(t) = e^{-\lambda t} \text{ mit } \lambda = \sum_{n=1}^{n} \lambda_i$$

Parallelschaltung:

$$F_{ges}(t) = \prod_{i=1}^{n} F_i(t) \text{ also } R_{ges}(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t))$$

Mehrheitsentscheidung bei Redundanz durch mehrere Komponenten:

$$R_{ges} = \sum_{i=k}^{n} \binom{n}{i} R(t)^i (1 - R(t))^{n-i}$$

4 Testen

Verifikation: Formaler Nachweis von Korrektheit und Einhalten der Spezifikation.
Testen: Ausführen verschiedener Testfälle und Überprüfung des Ergebnisses. Vollständigkeit nahezu nie erreichbar.

4.1 Bugs

- **Bohrbug:** Reproduzierbar; tritt immer unter den gleichen Umständen auf.
- **Heisenbug:** Tritt beim Versuch den Fehler zu analysieren nicht mehr auf.
- **Schrödinbug:** Wirkt sich nicht auf die Ausführung aus und lässt sich nur durch Codeanalyse finden.

4.2 Testen

Extremfälle und ungültige Eingaben sollten immer auch getestet werden (vor allem bei Black Box).
Beim White Box-Testing gibt es Statement/Line Coverage, Branch Coverage, Path Coverage, Multicondition/Predicate Coverage und Loop Coverage (jede Schleife mind. zwei mal).

4.3 Verifikation

Man kann den Code als endlichen Automaten abstrahieren (Kirpke-Struktur).
Es gibt folgende Quantoren [Computation Tree Logic: CTL] (A,E Pfadquantoren; Rest Zustandsquantoren):

A	für jeden Pfad
E	es gibt einen Pfad
Xp	im nächsten Zustand gilt
Fp	irgendwann in der Zukunft gilt
Gp	es gilt in Zukunft immer
pUq	es gilt p bis q gilt