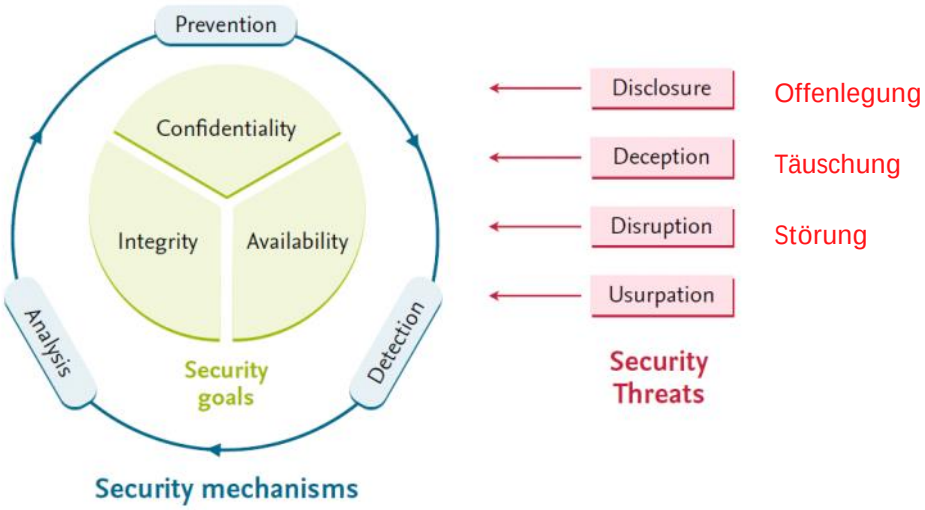
IT-Sicherheit

Freitag, 11. März 2022 14:14

Grundlegende Konzepte:



Offenlegung

Täuschung

Störung

- Ziele (CIA): unbefugt = unauthorized

	Confidentiality (Vertraulichkeit)	Integrity (Integrität)	Availability (Verfügbarkeit)
Ziel	Schutz vor unbefugter Offenlegung von Ressourcen (Daten/Gesprächen/etc.)	Schutz vor unbefugter Manipulation von Ressourcen	Schutz vor unbefugter Störung der Ressourcen
Schutz	- Verschlüsselung von Daten - Verstecken von Daten	- Autorisation - Checksums - Digitale Fingerprints	- Einschränkung - Redundanz - Vielfalt
Attacken	- Mithören eines Telefonats - Hacker liest Emails mit	- Ändern der Empfänger Adresse bei einer Banktransaktion - Daten manipulieren auf dem Computer/Handy	- Ddos Angriff auf eine Webseite - Formatieren der Hard Disk

- Definitionen:
  - Security:
    - Schutz gegen absichtliche/menschengemachte Threats (Bsp: Hacker, Ransomware)

bedrohungen

  - Safety:
    - Schutz vor versehentlichen Threats (Bsp: Blitzeinschlag, Erdbeben)
  - Threats (Threatklassen):
    - potentieller Verstoß gegen ein Ziel
    - Disclosure: unbefugter Zugriff auf Informationen
    - Deception: Akzeptanz falscher Daten
    - Disruption: Unterbrechung oder Verhinderung des ordnungsgemäßen Betriebs
    - Usurpation: unbefugte Kontrolle über Ressourcen
  - Attacke:
    - Versuch ein Sicherheitsziel zu verletzen (absichtlicher Threat)
    - Meistens Kombination mehrerer Threatklassen

- Sicherheitsmechanismen:
  - Policy:
    - Statement was ist erlaubt und was nicht
  - Mechanismus:
    - Methode oder Werkzeug zur Durchsetzung einer Sicherheitsrichtlinie
  - Strategien:

openssl snort wireshark

	Prevention	*Detection*	*Analysis*
Ziel	Verhindern von Angriffen **vor** der Verletzung von Sicherheitszielen	Erkennung von Angriffen **bei** der Verletzung von Sicherheitszielen	Analyse von Angriffen **nach** Verletzung von Sicherheitszielen
Beispiel	- Authentifizierung - Verschlüsselung - Beschränkung des Zugangs zu Informationen/Ressourcen	- Anti-Virus Scanner - Erkennung von bösartigen Code	- Computer Forensik - Untersuchung und Analyse von SIcherheitsvorfällen
Limitationen	- Oft nicht anwendbar (Bsp: bei offenen DIensten)	- Unbekannte Angriffe - Unsichtbare Angriffe	- Schaden könnte bereits entstanden sein

Einschränkung

- Weitere Konzepte:
  - Authentizität:
    - Vertrauenswürdigkeit von Informationen und Ressourcen
  - Accountability (Verantwortlichkeit):
    - Verbindung von Aktionen an User
  - Privacy:
    - Sicherheit und Kontrolle von persönlichen Informationen

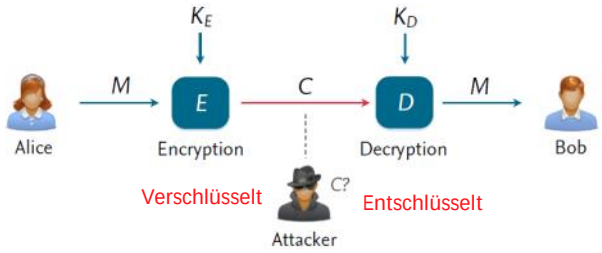
K2

Symmetric-Key Cryptography:

- Cryptography:
  - Wissenschaft von Informationssicherheit
  - Schutz der Vertraulichkeit und Integrität

- Cryptoanalysis:
  - Studieren von Attacken gegen Cryptography

- Cryptosystem:



Verschlüsselt Entschlüsselt

Cryptographic system for en/decrypting messages

- M = plaintext message C = ciphertext message
- K_E = encryption key K_D = decryption key

  - Cipher:
    - Encryption und Decryption Funktionen
    - E(M, K_E) = C and D(C, K_D) = M

  - Keyspace:
    - Menge von allen möglichen Schlüsseln

  - Symmetric-key cryptography:
    - K_E = K_D

- Was macht einen Cipher stark?
  - Confusion:
    - Komplexe Beziehung zwischen Key und Plaintext/Ciphertext
    - Aus M und C K zu bekommen sollte schwer sein
  - Diffusion:
    - Komplexe Beziehung zwischen Plaintext und Ciphertext
    - Aus C M zu bekommen sollte schwer sein

key schwer herauszufinden

Plain text schwer zu entschlüsseln

- **Kerckhoffs's Principle:**
  - Annahme, dass der Cipher vom Angreifer bekannt ist
  - Sicherheit sollte nur vom Key abhängen
  - --> Keyspace muss sehr groß sein

Key size	My Laptop	1 Million Cores
32 bit	6 seconds	0 seconds
64 bit	850 years	7 hours
128 bit	10^{22} years	10^{16} years
256 bit	10^{59} years	10^{52} years

Some Attack Types Einige Angriffsarten

- Klassische Angriffe:
  - Ciphertext-only
    - Angriffe nur auf den Ciphertext C
    - Man sieht nur C
  - Known-plaintext/Chosen-plaintext
    - Angreifer kennt M oder kann sich M aussuchen
    - Rückschlüsse auf den Key --> Zukünftige Nachrichten können auch gelesen werden

- "Dumme Angriffe":
  - Brute-Force
    - Alle Schlüssel raten
  - Purchase-key:
    - Gewalt, Geld, Etc einsetzen, um Schlüssel zu bekommen

vingenere cipher
zeile : abcde...
spalte : abcde..

- Security Level von Kryptosystemen:
  - Unconditionally secure (absolut sicher):
    - Nicht zu brechen mit unendlichen Ressourcen und allen möglichen Angriffen
  - Computationally secure (praktisch sicher):
    - Nicht zu brechen mit verfügbaren Ressourcen und bekannten Angriffen
    - Angreifer sind beschränkt durch Ressourcen (Comptation power, Zeit, Data)

Modell eines Angreifers, der Ressourcen nutzt

monoalpha
KEYABCDF.....

- Klassische Cipher:
  - Simple substitution ciphers
    - Rotiere Alphabet um k Buchstaben
    - Bsp: Caeser cipher, ROT13
    - Keyspace ist sehr sehr klein!

ROT13 keyspace : 1

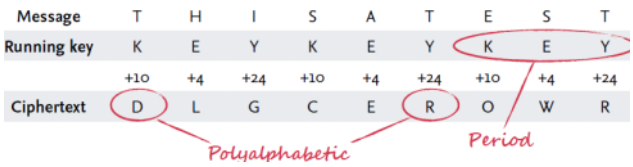
  - **Monoalphabetic substitution ciphers:**
    - Permutiere Buchstaben aus einen Alphabet
    - Keyspace schon deutlich größer
    - Buchstaben Frequenzen bleiben erhalten

Anzahl von Buchstaben k : k! keyspace

swach gegen analytical attack

huruf keberapa -1

  - **Vigenère Cipher:**
    - polyalphabetic substitution cipher
    - Kombination von mehreren simple Substitution ciphers
    - Rotation durch ein Word (key)
    - Einfach zu brechen



  - **One-Time Pad:**
    - XOR Ciphers
    - Bedingungen:
      - Key Länge gleich der Message Länge
      - Key bits are wirklich zufällig (kein pseudo zufall)
      - Key wird nur einmal benutzt und dann zerstört
    - Sicherheit:
      - **Unconditionally secure**
      - Durch den Zufall der Key Bits ist es unmöglich aus C M zu bekommen, denn jedes mögliche Nachricht könnte M sein, da alle Bits zufällig sind
    - Probleme:
      - Key Austausch schwer
      - Echter Zufall nötig
      - Heute nicht praktikabel

- $E(M, K) = M \oplus K = C$
- $D(C, K) = C \oplus K = M$

Why does this work?

$M \oplus (K \oplus K) = M \oplus 0 = M$

Example: $C = 01$

- $M = 00$ for $K = 01$, $M = 01$ for $K = 00$
  $M = 10$ for $K = 11$, $M = 11$ for $K = 10$

- Moderne Cipher:
  - Komplexe Substitutionen und Permutationen
  - Rundenbasierte Encryption und Decryption Algorithmen
  - Security vs. Effizienz

  - Block Cipher:
    - Encryption und Decryption von Blöcken fester Größe (oft 64 oder 128 bit)
    - Beispiel: AES, Serpent, Twofish, IDEA
    - Kurze Nachrichten werden aufgefüllt (Padding)
    - Lange Nachrichten werden aufgeteilt
    - Unterschiedliche Modi: ECB, CBC, OFB, CTR, …

Beispiel

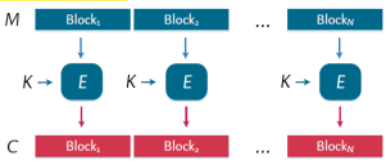
    - **AES:** Advanced Encryption Standard
      - Sicher und Effizient in Soft und Hardware (Computationally secure)
      - Substitutions und Permutations Netzwerk (S-Box & P-Box)
      - Block Cipher mit 128 bit
      - Key-size: 128, 192, 256 bits
      - Rounds: 10, 12, 14 (depending on key size)

    - Modi: Block cipher modes

1 ☐ Electronic Code Book (ECB):

- Strukturen in der Datei bleiben Erhalten
- Blöcke mit gleicher Eingabe werden auch gleich Verschlüsselt
- Super unsicher (Known-Plaintext Angriffe, Replay Angriffe)

same key for each block



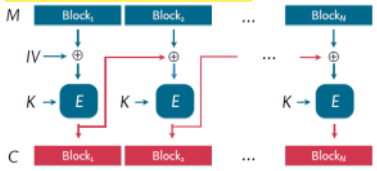
2 ☐ Cipher-Block Chaining (CBC):

- Verkettung der Cipher Blocks mit XOR Operator
- Resistent gegen Angriffe
- Fehlerfortpflanzung, wenn in einem Block etwas kaputt geht

error propogation

- 1 Bit Änderung/Fehler im ersten Block führt zu 50% Änderung im letztem Block

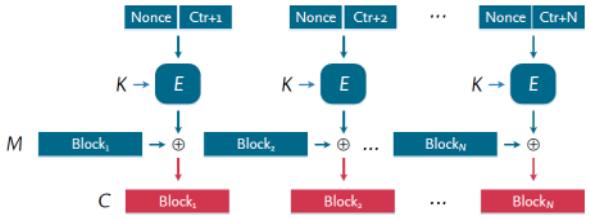
IV = random or unique Initialization Vector



3 □ Counter Mode (CTR):

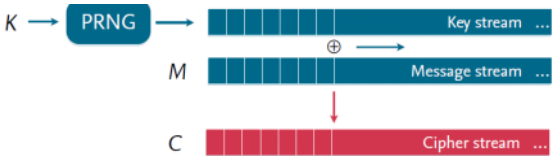
- Block Cipher wird wie eine Stream Cipher genutzt
- Zufälliger Zugriff auf Blocks
- Auch bei Fehlern in einem Block kann man einfach den kaputten Block ersetzen
- Nonce = Number used only once

same key but different result because of noncce and CTR



○ Stream Cipher: stromchiffre

- Bitweise Encryption und Decryption
- Verwendet einen Pseudo-Zufalls-Generator (PRNG)
- Sicherheit hängt von PRNG ab



Beispiel ▪ **RC4 Cipher:**

- □ Key size: 40 to 256 bits
- □ Key-scheduling Algorithmus initialisiert Substitutions S-Box
- □ Keystream wird durch das swappen von elementen in der S-Box berechnet

Summary

Kryptographie („Informationen sicher aufbewahren“)•Ver- und Entschlüsselung von Nachrichten

• Sicherheit sollte vom Schlüssel abhängen, nicht vom Algorithmus

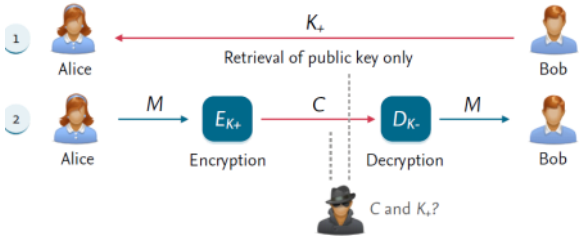
(Symmetrische Kryptosysteme)
• Sender und Empfänger verwenden denselben Schlüssel
• Verschiedene Verschlüsselungstypen (Block- und Stromchiffre)
• Funktionsweise abhängig von der App

K3 **Public-Key (Asymmetric) Cryptography:**

Problem * • Key-Exchange (Schlüsselaustausch)
- Sicherer Key-Exchange um sichere Kommunikation zu haben
- Kommunikation zwischen Unbekannten nicht möglich

* • Problem bei Symmetrischen Keys bei n Parteien: (n^2-n)/2 Schlüssel notwendig
Denn jedes Paar brauch eigenen Schlüssel

* • Lösung: zwei Schlüsselarten einführen
- ○ Public Key K+: zum verschlüsseln
- ○ Private Key K-: zum entschlüsseln
- ○ Schwer den Private-Key vom Public Key herzuleiten
- ○ Vergleiche Briefkasten: Alle wissen wo der Briefkasten steht und können was reinschmeißen, aber nur du hast den Schlüssel, um die Nachrichten zu öffnen



- Skalierbare Kommunikation mit n Parteien n= multiple
  - ○ Lineare Anzahl an Keys nötig: n Parteien = n Keys
  - ○ Alle Public Keys in einer DB speichern

Part2 • Mathematische Grundlagen:

- ○ Trapdoor one-way functions: F(x)=y
  - ▪ Mit **x** kann man leicht **y** berechnen
  - ▪ Mit **y** ist es schwer **x** zu erhalten
  - ▪ Mit **y** und einem **secret** ist es einfach **x** zu erhalten
  - ▪ Encryption mit Public Key - Berechne y
  - ▪ Decryption mit Private Key - Berechne x mit secret
- ○ Zahlen a und b sind co-prime, wenn gcd(a,b)=1
- ○ a*a^-1 = 1 (mod m)
- ○ Inverses existiert, wenn a und m co-prime
- ○ Zwei Arten von Trapdoor one-way funtions:

beispiel ▪ *Integer factorization:*

- □ Gegeben ein Integer n, finde seine m Primfaktoren
- □ Bsp: n = 4711 --> p_1=7, p_2=673 --> 7*673=4711
- □ Schwer die Primfaktoren zu finden/berechnen

Trapdoor

• Problem schwer zu lösen, aber leicht zu überprüfen (Asymmetrie)
• Kein polynomieller Algorithmus zur Lösung bekannt

Beispiel
- *Diskreter Logarithmus:*
  - Gegeben g,p,b, finde integer a, sodass:
  - g^a=b (mod p)
  - Schwer a zu finden/berechnen

Asymmetric Algorithms

Part3

- RSA Algorithmus: Rivest-Shamir-Adleman algorithm
  - Basiert auf Integer factorization und Verschlüsselung und Signierung
  - Standard Algorithmus für Public-key Cryptography
  - **Key generation**
    Choose random primes p, q and compute $n = p \cdot q$
    Compute Euler function $\varphi(n) = (p-1)(q-1)$
    Choose random encryption key e with $\gcd(e, \varphi(n)) = 1$
    Compute decryption key $d = e^{-1} \bmod \varphi(n)$
  - **Encryption with public key e, n**
    Encrypt message m to ciphertext $c = m^e \bmod n$
  - **Decryption with private key d**
    Decrypt ciphertext c to message $m = c^d \bmod n$
  - Sicherheit hängt von der Größe der Primzahlen ab
  - --> Keys mit 4096 bits und mehr vergewissern Sicherheit

Part4

- Diffie-Hellman Key Exchange:
  - Basiert auf Diskreten Logarithmus
  - Nur Key-Exchange, **keine** Verschlüsselung!!!
  - **Initialization**
    Alice and Bob agree on prime n and generator g
    For all $0 < b < n$ there is an x such that $g^x \bmod n = b$
  - **Generation of secrets**
    Alice select random number x and sets $X = g^x \bmod n$
    Bob selects random number y and sets $Y = g^y \bmod n$
  - **Key exchange**
    Alice sends X to Bob and Bob sends Y to Alice
    Alice computes shared key $k = Y^x \bmod n$
    Bob computes shared key $k = X^y \bmod n$
  - Sicherheit hängt von der Größe von X und Y ab

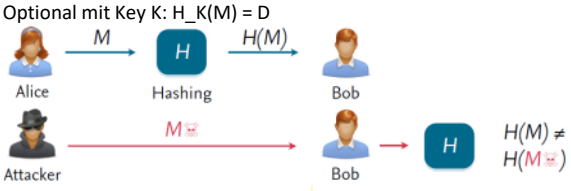
Summary

•Sicherheit von Algorithmen mit symmetrischen Schlüsseln ⇝ Komplexität
- Diffusion und Verwirrung durch komplizierte Bitoperationen
• Sicherheit von Algorithmen mit asymmetrischen Schlüsseln ⇝ Schwere Probleme
- Falltüreigenschaft(Trapdoor) basierend auf schweren mathematischen Problemen
- Bis heute keine polynomialen Lösungen bekannt
• Bleiben diese mathematischen Probleme schwer?
- … Fortschritte im Quantencomputing
- … neuartige polynomiale Algorithmen (oder sogar: P = NP?)

• Public-Key-Kryptographie
- Asymmetrische Schlüssel → sicherer Schlüsselaustausch
- Skalierbare Verschlüsselung mit mehreren Parteien
• Sicherheit basierend auf Trapdoor-Einwegfunktionen
- Integer-Faktorisierung → RSA-Algorithmus
- Diskreter Logarithmus → Diffie-Hellman-Schlüsselaustausch
- Sicherheit hängt von großen Schlüsselgrößen ab (> 3000 Bit

K4

Hybrid Cryptosystems:

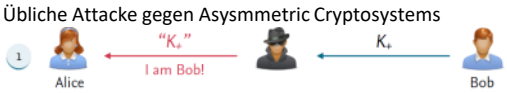
- Hashfunktionen:
  - Auch genannt message digest oder fingerprint
  - H(M) = D
  - Hashes eine Nachricht M zu einer festen Anzahl an Bits
  - One-way Eigenschaft:
    - Mit M ist es einfach H(M) zu berechnen
    - Mit gehashtem Wert D schwer M mit H(M) = D zu finden
  - Optional mit Key K: H_K(M) = D
  - 

  - Anwendung:
    - Checksum für große Dateien (Bsp: ISO Images, Filme, etc)
  - Kollisionen
    - Schwache Kollisionsresistenz:
      - Unmöglich M' mit M ungleich M' und H(M) = H(M') zu finden
      - Geburtstagattacke möglich
    - Starke Kollisionsresistenz:
      - Unmöglich ein Paar M und M' zu finden für die gilt:
        - M ungleich M' und H(M) = H(M')
      - Geburtstagattacke nicht mehr möglich
  - SHA2 (Secure Hash Algorithm v2):

- Übliche Hash Größen: 256 und 512 bits
- Runden: 64 und 80 je nach Hash Größe
- Netzwerk aus arithmetischen und logischen Operationen

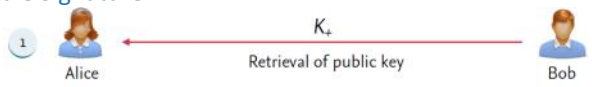
MITM (Man-in-the-Middle):

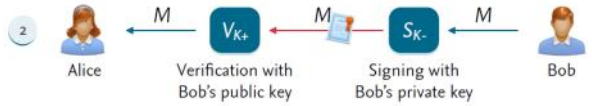
- Übliche Attacke gegen Asysmmetric Cryptosystems

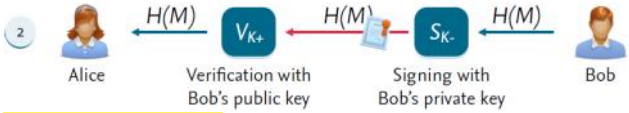


  - Bob sendet public key an Alice
  - ABER: Angreifer sitzt dazwischen und erhält Bobs public key und sendet dann seinen public key an Alice
  - Jetzt denkt Alice dass sie mit Bob kommuniziert, aber eigentlich kommuniziert sie mit dem Angreifer
  - Der Angreifer kann jetzt alle Nachrichten von Alice empfangen und entschlüsseln, da Alice ihre Nachrichten mit dem public key des Angreifers verschlüsselt hat
  - Und der Angreifer kann jetzt die Nachricht (manipuliert) an Bob zurücksenden, damit er denkt, dass alles okay ist
  - Also der Angreifer hat die gesamte Kontrolle über den Nachrichtenverlauf und kann im Namen von Alice Nachrichten an Bob senden, die sie gar nicht geschrieben hat
- Schutz vor MITM:
  - Manueller Vergleich der public keys mit Hash Values
  - Speicher der bestätigten public keys in einer DB
  - Voraussetzung: Sicherer Austausch der Fingerprints beim ersten Verbinden

Digitale Signaturen:





- Signing: Encryption mit private Key K
- Verification: Decryption mit public key K
- Besser: Signieren mit Hashing:
  - 

  - Signiere H(M) statt nur M
- Signaturen und RSA
- Signing mit private key d: s = H(M)^d mod n
- Verification mit public key e: H(M) = s^e mod n

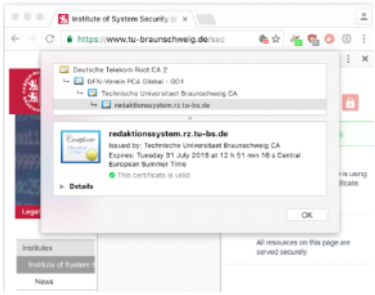
- Problem:
  - Public keys nicht verbunden mit der Identität der User
- Lösung:
  - Validieren und Signieren der Public keys von einer dritten vertrauenswürdigen Partei (Bsp: von der Uni/Unternehmen)
  - Verteidigung gegen MITM Angriffe

Hybrid Cryptosystems:

- Problem:
  - Public key cryptography ist sehr rechenintensiv
- Lösung:
  - Kombination aus asymmetrischen und symmetrischen Kryptosystemen
    - Public-key Algorithmen für den sicheren Austausch eines session key
    - Symmetrische Block cipher für die effiziente (dennoch sichere) Verschlüsselung
    - --> Das Beste aus beiden Systemen sichere und effiziente Kommunikation
- Forward secrecy:
  - Problem: wenn gleicher Schlüssel lange bleibt
    - Angreifer zeichnet den gesamten Nachrichten Verlauf auf, kann sie aber noch nicht entschlüsseln
    - Aber nach vielen Jahren und neuen Schwachstellen evtl möglich
    - Dann kann der Angreifer jedoch rückwirkend den gesamten Nachrichten entschlüsseln und lesen
  - Kurzlebige session keys vs. Langlebige personal keys
  - Ständiger Wechsel der langlebigen keys nicht möglich
  - Spezielles Design für Hybride Kryptosysteme
  - Beispiel:
    - Forward secrecy mit Diffie-Hellman und RSA
      - Key-Austausch mit DH
      - Zufällige Initialisierung von Schlüsselpaaren für jede Session
      - Austausch wird signiert mit langlebigen RSA keys

Public Key Infrastructure (PKI):

- Management von Vertrauen durch public key cryptography

- Signaturen von Keys, Attributen, etc.
- Certificate Authorities (CA) agieren als vertrauenswürdigen Parteien
- "Chain of trust" mit mehreren Schichten
- 

○ SSL/TLS (Secure Sockets Layer/Transport Layer Security):

- Standard Protokoll für sichere Kommunikations
- Peer-to-peer encryption
- Hybrides Kryptosystem mit mehreren Algorithmen
- Unterstützt digitale Zertifikate (X.509 Standard)
- Einfach zu integrieren
- Im Web: HTTPS (S für SSL)
- Nur so sicher wie die Vertrauenskette
  - □ Angreifer greifen die Signatierungsstellen an und haben dann die Kontrolle über die Signaturen/Zertifikaten
- Lösung:
  - □ Certificate pinning
    - ◆ Beschränkung der Vertrauenskette auf bestimmte gepinnte Zertifikate
    - ◆ Pinning voon Wurzel, Mittel oder Endzertifikat möglich
    - ◆ Bsp: HTTP public key pinning in Chrome und Firefox
  - □ Certificate transparency:
    - ◆ Tracking von neuen Zertfikaten in öffentlichen, append-only logs
    - ◆ Ständiges Überwachen beim Hinzufügen neuer third-partie Zertifikaten

○ PGP (Pretty Good Privacy):

- Weit verbreitet bei der Signierung und Verschlüsselung von Emails
- Hybrides Kryptosystem mit mehreren Algorithmen
- Einfache Integration in vielen Email Clients
- In die Jahre gekommen --> nicht mehr so sicher
- Probleme:
  - □ EFAIL: Attacke gegen OpenPGP
    - ◆ Verschlüsselte Daten in einer anderen Email packen und zum Opfer senden
    - ◆ Email Client entschlüsselt dann die Daten
    - ◆ Entschlüsselte Daten werden zurück zum Angreifer gesendet (Bsp: via Link)
  - □ Angriffe, die Signaturen von OpenPGP fälschen

• Public-Key-Kryptographie
- Man-in-the-Middle-Angriffe und -Abwehr
- Digitale Signatur und Verifizierung von Daten
• Implementierungen
- Hybride Kryptosysteme – symmetrisch und asymmetrisch
- Public-Key-Infrastrukturen (PKI) zur Verwaltung von Vertrauen
- Beliebte Implementierungen: SSL und PGP

Kap 5

Authentication und Authorization:

- Authentication = Binde eine Identität an ein Subjekt (Nutzerkonto, Bsp: y-Nummer ist an Identität des Studenten gebunden)
- Bestätigung der Identität durch:

	Knowledge Factors	Ownership Factors	Human Factors
Ziel	Subjekt bestätigt Identität mit Wissen	Subjekt bestätigt Identität durch den Besitz von etwas	Subjekt bestätigt Identität durch menschliche Eigenschaften
Beispiel	Passwort	Yubikey (physisches Gerät)	Biometrische Daten: Fingerabdruck, Iris
Vorteil	- Einfach umzusetzen - Ersetzen einfach	- Schwer zu klonen/reproduzieren - Einfach Handhabung	- Ideale Verbindung zwischen Identität und Subjekt - Von Natur aus gegeben
Nachteil	- Mensch kann sich nicht alle Passwörter merken - Vertraulichkeit nicht gewährleistet, wenn andere Passwörter haben	- Verlieren/Stehlen des Geräts problematisch - Oft extra Hardware benötigt	- Austauschen schwierig (Privacy Problem) - Immer wahre Identität preisgeben, keine Anonymität mehr möglich

- Multi-Factor Authentication:
  - ○ Verbindung mehrerer Faktoren (2-Faktor-Auth)
  - ○ Bsp: Bezahlen mit Karte:
    - Ownership, der Besitz der Karte
    - Knowledge, PIN

- Passwörter:
  - ○ Information, die eine Identität bestätigt
  - ○ Oft kombiniert mit (a)symmetric Key cryptography
  - ○ Probleme:
    - Password Snooping/Sniffing:
      - □ Passwort wird unverschlüsselt übertragen, Angreifer kann Passwort einfach lesen
      - □ Malware auf dem PC liest Eingaben mit und hat dann das Passwort
    - Password Guessing (online) oder cracking (offline):
      - □ Dictionary Attacks: Raten mit riesigen Wörterbüchern von bekannten Passwörtern
      - □ Brute-force Attacks: Raten aller möglichen Passwörter

e.g. password is mapped to key of symmetric cipher, password protects private key of public-key algorithm

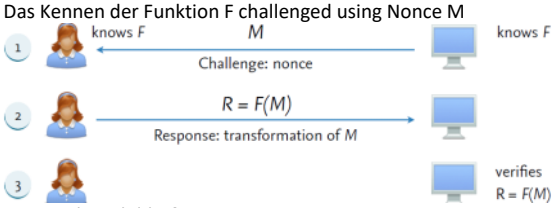
        - Menschliche "Dummheit":
            - Schwache Passwörter
            - Überall dasselbe Passwort

    - Passwort Speicher:
        - Passwörter sollten niemals im Klartext gespeichert werden
        - Nur hashed Passwörter speichern
            - Bsp: hashed_pw = hash(password)
            - Einfach zu verifizieren: hash(input) == hashed_pw
            - Schwer das Passwort vom hashed Passwort zu bekommen
        - Besser: Salted Passwords
            - Bsp: hashed_pw = hash(password+salt)
            - Salt (zufälliger String) wird in Klartext gespeichert [salt, hashed_pw]
            - Vorteil:
                - Angreifer kann ein einzelnes Passwort cracken, ABER es ist nicht mehr mögliche alle Passwörter einer DB zu cracken
            - Sicherheit hängt von der Qualität des Passworts, des Hashs und des Salt ab
        - Slowing-down Crackers (Key stretching):
            - Passwörter werden mehrmals gehashed --> Dauert länger
            - Bsp: statt 1000 Versuche pro Sekunde, dann nur noch 1 Versuch pro Sekunde
            - Dadurch brauchen Angreifer viel viel länger beim Passwort cracking

    - Gute Passwörter:
        - Passwörter sollten sehr schwer zu raten sein
        - Keine Dictionary words, Namen, Daten, Mustern
        - Minimum Länge und am besten zufällig
        - Beste Lösung: Passwort Manager
            - ABER nicht online, sondern DB liegt privat bei einem selbst

    - One-Time Passwords:
        - Passwörter werden über die Zeit schlechter
            - Angreifer haben mehr Zeit das Passwort zu cracken
        - Idee: Passwort wird immer nur genau einmal verwendet
            - Bsp: TAN Listen

    - RSA SecureID:
        - 2-Faktor-Auth
        - Knowledge und Ownership
        - Ownership generiert alle 60s einen neuen Code
        - Anmelden mit Passwort und Code
        - Code Generation:
            - Gerät wird initialisiert mit random seed/number
            - Code wird generiert durch seed und current time
                - --> One time pwd

    - Challenge-Response Authentication:
        - System und User teilen sich eine geheime Funktion F
        - Das Kennen der Funktion F challenged using Nonce M
        - 

        - Wie wird F gebildet?
            - F = H(M+P) - Hash Funktion H und Passwort P
            - F = E_P(M) - Encryption Funktion E und Passwort P
            - Schwer P zu finden, wenn F stark ist
        - Methoden:
            - One-time Passwörter:
                - Challenge(index of password); response(password)
            - RSA SecureID:
                - Challenge(current time); response(authentication code)

anzahl pass = erlaubte ^(L

- Access Control:
    - Kontrolliert, was ein Subjekt machen darf
    - Management von Zugriffsrechten
    - Oft mit Authentication verbunden
    - Bsp: Read, Write, Exec Dateien
    - Access Control Matrix:

Objects

Subjects	File 1	File 2	Process 1	Process 2
User 1	read		control, send	receive
User 2	write	read	send	
User 3	read, execute	read	control	control

Access control lists — Capabilities

- Access Control ist nicht einfach in der Realität
- Charakteristiken:
  - Definition von Objekten und Subjekten
    - Bsp: Subjekte können User, Prozesse oder Host sein
  - Repräsentation von Rechten:
    - Spalten (Access Control List/ACL)
      - Speichern der Rechte mit dem Objekt
      - Effiziente und dezentrale Verwaltung der Rechte
      - Liste der Subjekte, die zu einem Objekt Rechte haben kann sehr groß werden
      - Bsp: OpenBSD packet filter:
        - Verbiete Zugriff auf SSH von jedem Host
    - Zeilen (Capabilities):
      - Speichern der Rechte an das Subjekt
      - Speichern der Rechte eines Subjekts einfach
      - Feine Rechte schwer zu implementieren
      - Bsp: Linux Capabilities:
        - Beschränke Rechte um das System neu zu starten
  - Management von Rechten:
    - Discretionary Access Control (DAC):
      - Besitzer eines Objekts kontrolliert die Rechte
      - Unsicher, wenn es einen Benutzerwechsel gibt
    - Mandatory Access Control (MAC):
      - Von vorne rein alle Rechte festgelegt
      - Keine Änderung möglich
      - Bsp: Space Shuttle
      - Sehr sicher, aber mühsam zu bedienen und zu entwerfen
    - Role-based Access Control (RBAC):
      - System erzwingt Zugriffsrechte über Rollen/Gruppen
      - Zwischen DAC und MAC

kap 6

Network Attacks and Defenses:

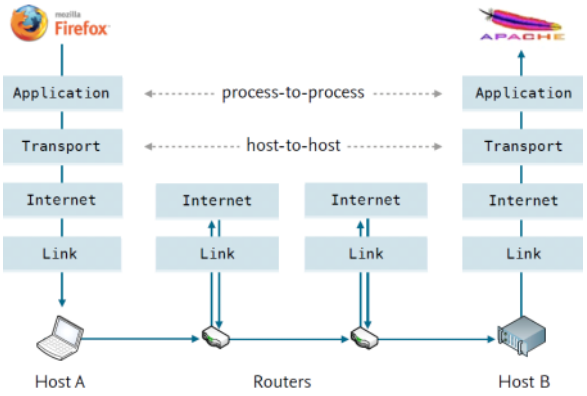
- Layers:
  - Theory-driven model: **OSI model** (ISO, 1983)

7 Application	Application
6 Presentation	
5 Session	
4 Transport	Transport
3 Network	Internet
2 Data link	Link
1 Physical	

  - Practice-driven model: **TCP/IP model** (DARPA ~70s)
- TCP/IP:

Layers	Functions	Examples
Application	Interfacing with network applications	HTTP, FTP
Transport	Delivery and multiplexing of data to network applications	TCP, UDP
Internet	Addressing and transfer of data between hosts and gateways	IP, ICMP
Link	Interfacing with and control of physical devices	PPP, ARP

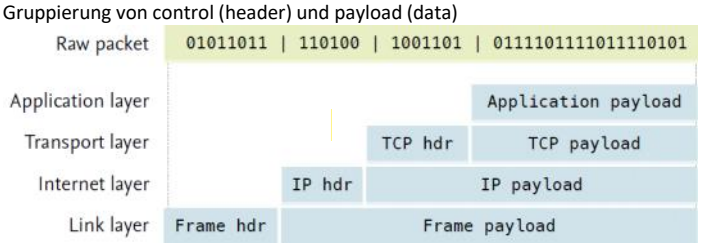
TCP vs. UDP (TCP for reliable communication, UDP for fast, connectionless communication)



CIDR Subnet Mask # of Hosts (Usable)
/24 255.255.255.0 254 (256 - 2)
/25 255.255.255.128 126 (128 - 2)
/26 255.255.255.192 62 (64 - 2)
/27 255.255.255.224 30 (32 - 2)
Why subtract 2 from the total host count?

1 address is reserved for the network (.0 in 192.168.1.0/24).
1 address is reserved for the broadcast (.255 in 192.168.1.0/24)

- Network Packets:
  - Computer Networks = packet-switched networks
  - Packete durch layers strukturiert
  - Gruppierung von control (header) und payload (data)



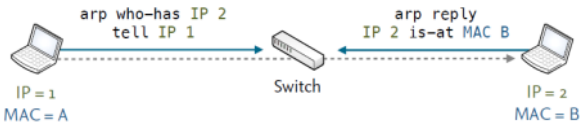
- Classic Network Attacks:
  - Für alle Layer möglich
  - Betrifft alle Sicherheitsziele
  - Warum?
    - Fehler in Protokollen oder Netzwerkdesign
    - Schwachstellen in Implementierungen
    - Falsche Konfiguration von Laien
    - Falsche Verwendung von Diensten
  - Attacken:
    - Spoofing:
      - Netzwerk Nachrichten mit gefälschten Daten
    - Hijacking:
      - Übernahme von Verbindungen und Sitzungen
    - Flooding:
      - DDOS Angriffe

spoofing refers to the act of impersonating a legitimate entity by falsifying data to gain unauthorized access to systems, steal sensitive information, or spread malware

  - Network Sniffing:

eavesdrop = abhören

    - Betrifft alle Layer
    - Ist das Abhören von Netzwerkpaketen
    - Physischer Zugriff auf Kommunikationsmedium (air, wire)
    - Passives und unbemerktes Abhören der Nachrichten
    - Automatisches Parsen von Protokollen in Paketen

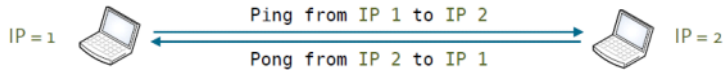


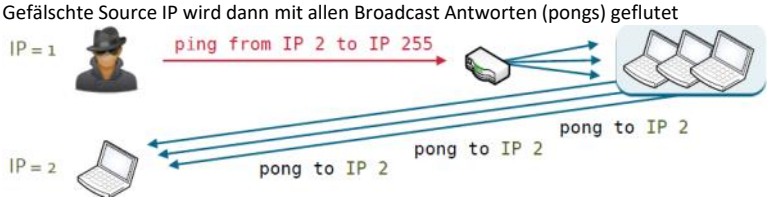
    - Schaden:
      - Keine richtige Attacke
      - Sondern verletzt besonders die Vertrauenswürdigkeit

vetraulichkeit

  - ARP Spoofing:
    - Betrifft den Link Layer
    - Address Resolution Protocol (ARP)
    - Standard Protokoll für den Link Layer
    - Mapping der IP zu einem Device (MAC)
    - Problem:
      - ARP antwortet mit gefälschter IP Adresse
      - Opfer sendet direkt den ganzen Traffic zum Angreifer (MITM)
    - Schaden:
      - Verletzt Vertrauenswürdigkeit , da der Angreifer alles mitlesen kann
      - Verletzt Integrität, da der Angreifer Daten manipulieren kann

  - Smurf Attacks:
    - Betrifft Internet Layer
    - Broadcast Nachricht an das gesamte Subnetz
    - ICMP (Internet Control Messages Protocol)
      - Kontrolliert Nachrichten

- Bsp: Ping
- 

  - Angriff:
    - Flutet den Cache mit gefälschten Broadcast Ping Nachrichten
    - Fälscht Source IP vom ICMP echo (ping) request
    - Gefälschte Source IP wird dann mit allen Broadcast Antworten (pongs) geflutet
    - 

  - Schaden:
    - Verletzt die Verfügbarkeit der Netzwerk Bandbreite des Opfers

- Amplification Attacks:
  - DDOS Angriffe
  - Asymmetrie in aus und eingehenden Traffic
  - Smurf und Fraggle Angriffe

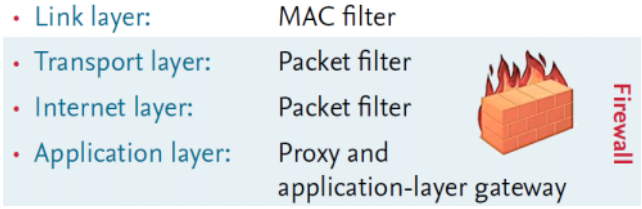
In information security, amplification attacks are a subset of Distributed Denial of Service (DDoS) attacks where an attacker exploits the functionality of network protocols to significantly increase the volume of traffic directed at a target system.

Network Defense:

- Grundlegende Konzepte:
  - Cryptography: Verschlüsseln und Verifizieren der Daten
  - Authentication: gegenseitige Authentifizierung der Parteien
  - Access Control: Beschränkungen der Zugriffsrechte und Kontrolle der Kommunikation

- Reaktive Sicherheitskonzepte:
  - Vulnerability assessment: Das Finden von Schwachstellen
  - Intrusion detection : Das Finden von Angriffen
  - Computer forensic : Das Finden von Angreifern

- Cryptography in Netzwerken:
  - Netzwerkprotokolle mit Sicherheitserweiterungen
    - Schutz der Vertrauenswürdigkeit und der Integrität
    - Auf alle Layer anwendbar
  - Symmetric-key Cryptography:
    - Effiziente Verschlüsselung und Verifikation von Netzwerkdaten
    - Verifikation mit Hashfunktionen
  - Public-key Cryptography:
    - Austausch von Session Keys
    - Signieren und Verifizieren der Keys und Daten

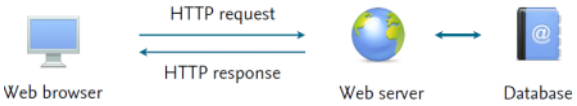
- Access Control in Netzwerken:
  - Oft ACLs auf Netzwerkobjekte (nets, hosts, ports)
  - Auf mehrere Schichten anwendbar
  - Realisiert durch Listen, Regeln und Filter
  - 

  - Firewalls:
    - Ein Host der den Zugang zum Netzwerk vermittelt
    - Überprüfen aller eingehenden und ausgehenden Pakete
    - Access Control auf mehreren Layern
    - Schutz der Netzwerkdienste vom inneren Netzwerk
      - Webserver darf nach außen (HTTP)
      - File Server nur im lokalem Netz (SMB)
    - Filtern von ausgehenden Datenverkehr:
      - Shell Verbindung zu anderen Hosts erlaubt
      - Chat Services werden blockiert
    - Design und Operation sind nicht einfach
      - Blocken von Traffic der nicht blockiert werden sollte
      - Filtern oft nur auf Application Layer möglich
  - Desktop Firewall:
    - Hostbasierte Variante der regulären Firewall
    - Blockiert ungewollten eingehenden Netzwerktraffic
    - Benachrichtig und warnt User vor ausgehenden Traffic
    - Monitors Anwendungen welche den Netzwerktraffic überwachen

- Attacks vs. Defense:
  - Rennen zwischen Angreifer und Vertediger
  - Ein Problem behoben, ein anderes hat sich geöffnet
  - Früher öfter Server angegriffen --> Heute mehr auf private Anwender

Kap 7

Web Security:

- Immer mehr Webanwendungen
- Client/Server Model mit HTTP Kommunikation
- Code beim Client: HTML, JavaScript, Flash,…
- Code beim Server: PHP,JSP,ASP,SQL,…
- Klassisches Beispiel:
  - 

- Sicherheitsprobleme:
  - Schlechte Cryptography, schlechte Authentifizierung
  - Schlechte und falsche Implementierungen
  - Spezifische Probleme bei Webanwendungen:
    - Client: Cross-site scripting
    - Server: Code injection und path traversal

  - SQL Injection:

A simple example: **password check in PHP**

```
$name = $_GET ["name"];
$password = $_GET ["password"];
$query = "SELECT * FROM users WHERE name = '$name'
                        AND password = '$password'";
```

- **Let's send another HTTP request …**

Condition is always true (password irrelevant)

```
$name = "steve";
$password = "x' or '1'='1";
$query = "SELECT * FROM users WHERE name = 'steve'
                    AND password = 'x' or '1'='1'";
```

- Auswirkungen des Angriffs:
  - Information leakage
  - Datenmanipulation
  - Code execution
- Warum ist das möglich?
  - Ungenügende Validierung der eingegebenen Daten!
- Gegenmaßnahmen:
  - Umgehen von Kontroll und Syntaxzeichen
  - Vorbereitete SQL statemnets
    - Keine Vermischung von Code und Daten mehr möglich machen

  - Remote Code execution:
    - Shell code injection
    - Beispiel:
      - Programm beim Server:
        -
```
<?php
   $data = $_GET ["data"];
   $output = shell_exec("cat "+ $data);
   echo "<pre>$output</pre>";
?>
```

      - HTTP Request des Angreifer:

Let's send another HTTP request …

        - 

      - Was passiert beim Server:

```php
<?php
   $data = "foo; rm -rf /";
   $output = shell_exec("cat foo; rm -rf /");
   echo "<pre>file not found</pre>";
?>
```

Execution of arbitrary shell commands

- Also es wird versucht alles im Dateisystem zu löschen

- Auswirkungen des Angriffs:
  - Der Code des Angreifers wird auf dem Zielserver ausgeführt
  - Angreifer hat die Berechtigungen wie der Betreiber der Webseite
  - Varianten mit PHP oder ASP
- Warum ist das möglich?
  - Wieder ungenügende Überprüfung der eingehenden Daten!!!
- Gegenmaßnahmen:
  - Keine Ausführung von Code von Webanwendungen erlauben
  - Umgehen von Steuer und Syntaxzeichen

- Path Traversal:
  - Wie Remote Code Execution
  - Angreifer versucht diesmal den Pfad im Zielsystem zu durchlaufen
  - Dadurch hat der Angreifer Zugriff auf Ordner, die er nicht haben sollte

Potential break out from directory of web application

- Auswirkungen des Angriffs:
  - Zugriff auf Dateien die außerhalb der Reichweite von Webanwendungen liegen
  - Potentieller Angriff auf Konfigurationsdateien und Source Code
  - Kombination mir remote code execution möglich und dann sehr gefährlich
- Warum ist das möglich?
  - Wieder ungenügende Überprüfung der eingehenden Daten!!!
- Gegenmaßnahmen:
  - Whitelisting von erlaubten Dateien und Ordnern
  - Normalisierung von Dateipfaden (meint ../ wirklich das was es soll?)

- Ursache der meisten Schwachstellen:
  - Dynamische Codeerzeugung durch String Operationen
  - Trennung von Code und Daten nicht erzwungen
  - Gefährliche Zeichen in Strings:
    - SQL injection: ' (termination of Strings)
    - Remote code execution: ; (termination of commands)
    - Path Traversal: ../ (move to upper directory)

Web Sessions (in HTTP):

- Sessions sind notwendig für online shopping etc
- Webanwendungen müssen Session Tracking iplementieren
- Session Ids tracken User über HTTP Requests hinweg
- 3 übliche Arten:
  - URL rewriting:
    - Session Ids automatisch an URL angefügt
    - Entwickler müssen alle URLs in Anwendungen fixen
    - Problem:
      - Session Id in HTTP Header sichtbar und wird somit geleakt
  - Form-based session Ids:
    - Session Id wird in versteckten Formfeldern gespeichert
    - Navigation der Anwendung muss Form benutzen
    - Transport der Session Id im Body von POST requuests
    - Problem:
      - Zurückbutton in Web-Browser

Wenn ein Benutzer den Zurück-Button betätigt, lädt der Browser die vorherige Seite aus seinem lokalen Cache. Dadurch kann es passieren, dass die Session-ID, die in einem versteckten Formularfeld gespeichert ist, nicht aktualisiert wird und einen veralteten Wert enthält. Dies kann zu Inkonsistenzen führen, da der Server die alte Session-ID möglicherweise nicht mehr als gültig erkennt, was zu Sitzungsfehlern oder unerwartetem Verhalten der Anwendung führen kann.

  - Cookies:
    - Persistenter Speicher vom Browser zur Verfügung gestellt
    - Datensatz mit HTTP-Headern oder scripting
    - Cookies werden automatisch zur Herkuntsdomain hinzugefügt
    - Beste Lösung

Um dieses Problem zu vermeiden, ist es gängige Praxis, Session-IDs in Cookies zu speichern. Cookies werden vom Browser automatisch bei jeder Anfrage mitgesendet und aktualisiert, wodurch die Konsistenz der Session-ID gewährleistet wird, selbst wenn der Benutzer den Zurück-Button verwendet. Allerdings müssen dabei Sicherheitsaspekte berücksichtigt werden, um Risiken wie Session-Hijacking zu minimieren.

- Sessions und Sicherheit:

- Beim Login Prozess wird Cookie erstellt
- Problem:
  - Sicherheitsrisiko, wenn die Session Id geleaked wird
  - Nur temporäres Problem, da Session Ids nicht ewig leben
- Sichere Session Ids:
  - Schwer zu raten --> lange zufällige Strings
  - Schwer mitzulesen --> Verschlüsselt übertragen
- Cross-site-scripting bleibt eine Gefahr

Client-side attacks:

JavaScript:

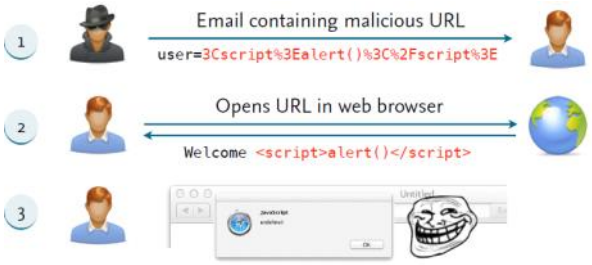
- Mächtiges scripting Interface
  - Zugriff auf URLs und Formfelder
  - Zugriff und Modifikation von Cookies
  - Ausführen von dynamischen Code
- Same-Origin Policy
  - Zugriff auf Elemente von der gleichen Origin beschränkt
  - Auch anwendbar auf Flash und CSS
  - Zwei Elemente haben die gleiche Origin, wenn…
    - Protokoll identisch
    - Host identisch
    - Port identisch
    - 

default port http : 80
https : 443
SSH : 22
DNS : 53

Cross-site Scripting (XSS):

- Injection von JavaScript Code
- Umgeht die Same-Origin Policy bypass = umgehen
- Ursache:
  - Anzeige von unzureichend validierten Daten
- Möglichkeiten mit XSS:
  - Gefährdung des Web-Browsers
  - Fälschung von Webinhalten
    - Angreifer manipuliert Inhalte auf der Webseite
  - Mitlesen von Logindialogen
    - Angreifer nutzt JavaScript, um Anmeldedaten zu bekommen
  - Session und Browser Hijacking:
    - Angreifer klaut Session Id und gibt sich als der echte User aus
  - Viele weitere Webangriffe

- Einstiegspunkte:
Injection von Tags:	<script>…
Ausbrechen von Attributen:	<img alt="…>
Javascript URLs:	javascript:…
Dateien die JavaScript enthalten	SVG

- Übliche Arten von XSS:
  - Reflected XSS (non-persistent):
    - Angreifer sendet Schadcode direkt zum User
    - Bösartiger Code wird vom Webserver reflektiert
    - 

  - Stored XSS (persistent):
    - Schadcode direkt beim Webserver
    - Benutzeranfragen lösen das Senden des bösartigen Code aus
    - 

  - DOM-based XSS:
    - ???

- Weird XSS:
  - Jede Art von Daten ist eine mögliche Sicherheitslücke für Angreifer
  - Bsp: XSS über SIP(Internet telephony)

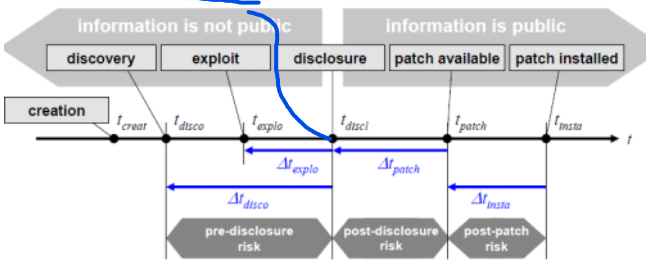
- Verhindern von XSS:
  - Validierung von allen Userdaten
  - Umgehen von GET und POST Parametern
  - Validierung von externen Daten (Bsp: DB)
  - Überprüfe sonstige Quellen (Bsp: Cookies, etc)
  - Umfassende Eingabevalidierung nicht möglich
    - Ausgangsdaten oft einfacher zu überprüfen
  - Allgemein:
    - Whitelisting ist dem Blacklisting vorzuziehen

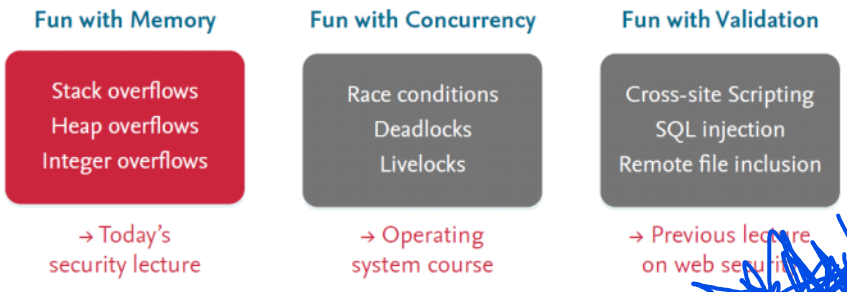
Vulnerabilities and Exploits:

- Grundlagen:
  - Vulnerability (Schwachstelle): exploitable flaw in a system or software
    - Ausnutzbare Schwachstelle/Sicherheitslücke in einem System oder einer Software
    - Grundlage für alle Angriffe gegen die drei Sicherheitsziele
    - Software bug nicht zwingend immer eine Schwachstelle
    - Woher kommen Schwachstellen?
      - Mängel im System Design
      - Implementierungsfehler
      - Falsche oder schlechte Konfiguration
      - Ungeeigneter Betrieb

    - Vorbeugen gegen Schwachstellen:
      - Sicherheitsbewusster Entwurf, Implementierung und Betrieb
      - Das braucht Zeit und Fachwissen --> teuer!!! example : practice of secure programming
        - Betriebe und User wollen aber alles günstig, also wird oft nicht viel Zeit und Wissen in die Entwicklung gesteckt

    - Häufige Ursachen:
      - Inkonsistenzen in der Semantik oder System bsp. schwachstelle in schnittstelle oder methode
      - Vermischung unterschiedlicher Kontrollstrukturen und Daten

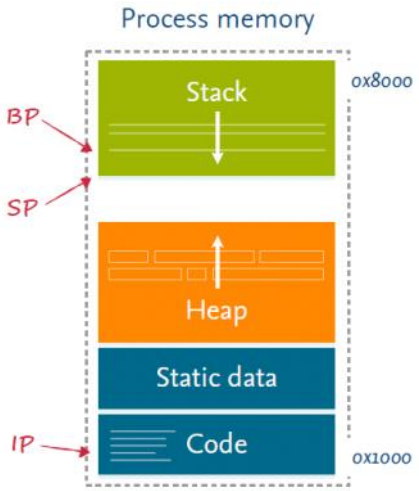
    - Schwachstellen Lebenszyklus:
      - Wichtigste Phase: Öffentliche Bekanntgabe der Schwachstelle
      - Vor Offenlegung als "zero-day" bezeichnet
      - 

    - Einige Schwachstellen Arten:
      - 

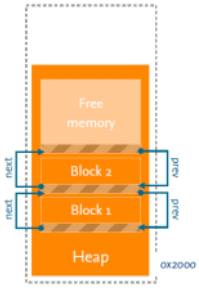
  - Exploits:
    - Ist ein Programm um eine bekannte Sicherheitslücke auszunutzen
    - Meistens Manipulation des Kontrollflusses
    - Umgehen von Authentifikation und Access Control
    - Proof of Concept vs. Schadsoftware
    - Übliche Exploit Phasen:
      - Injection von Code oder Daten
      - Manipulation des Kontrollflusses
      - Ausweitung der Privilegien Ausweitung = expansion

- Stack und Heap Speicher:
  - Programm Ausführung:
    - Lade Programm in den Speicher
    - Unterschiedliche Speichersegmente für unterschiedliche Datentypen
    - Spezielle Region für dynamische Daten
      - Stack (LIFO access)
      - Heap (Arbitrary access)
  - CPU Register:

PC/IP	Instruction Pointer	Zeigt auf aktuelle Anweisung
SP	Stack Pointer	Zeigt auf das Ende des Stacks
BP	Base Pointer	Zeigt zum aktuellem Stack Frame



  - Stack:
    - Temporärer Sepicher für Funktionen
    - LIFO access zu frames
    - Wächst von oben nach unten
  - Heap:
    - Allokation von beliebigen Blöcken
    - General-purpose Speicher
    - Wächst von unten nach oben
    - Verschiedene Heap Implementationen:
      - Meistens (double) linked list
      - Metadata am Start von jedem Block
    - Allokation:
      - Splitte Block vom freiem Speicher
    - Deallokation:
      - Merge Block mit freiem Speicher



  - Overflows:
    - Buffer Overflow:
      - Buffer = Array mit fester Länge (von einem Datentyp)
        - Stack: lokale Variable
        - Heap: Pointer zum Block
      - Buffer Overflow:
        - Schreibe über das Ende eines Buffers hinaus
        - Vorhandene Daten und Metadaten werden überschrieben
        - Geschriebene Daten sind vom Angreifer kontrolliert
      - Ausnutzbarkeit hängt von der Position im Speicher (Heap, Stack) und der CPU Architektur ab
      - Warum geht das?
        - Unzureichenden Speicher zugewiesen
        - Fehlendes Bewusstsein für das Überlaufproblem
        - Unbeschränkte String Funktionen: gets, strcpy, sprintf, …
        - Casting
        - Integer/Arithmetik Overflows
        - Gefährliche String Funktionen: strncpy, strncat, …
    - Stack Overflows:
      - Überschreit aktuellen und vorherige Stack Frames
      - Manipulation von lokalen Variablen
        - Umlenkung des Kontrollflusses durch Zuweisung von Variablen
      - … und Return Adressen
        - Injection von ausführbaren Code (Shellcode)
        - Umleitung der Retunr Adressen zum bösen Shellcode
    - Heap Overflow:
      - Überschreibt nächsten Heap Block und seine Metadaten
      - Manipulation von List Pointern
      - Sehr Implementierungsabhängig
      - Klassische Unlink Attacke:
        - Umleitung des Kontrollflusses
        - Angreifer überschreibt next und prev
        - Manipulation von den Daten auf dem Heap
        - Dadurch kann der Angreifer beliebige Daten an eine beliebige Adresse schreiben

Redirection of control flow by ...
• Manipulation of exception handlers
• Manipulation of maintenance functions
• Manipulation of data on the heap
• Difficulty: Exact memory addresses need to be known

- Speicher Verteidigung:
  - Statisches Messen:
    - Boundary Checks zur Kompilierzeit
    - Austauschen von unsicheren Funktionen
  - Dynamisches Messen:
    - Boundary Checks zur Laufzeit
    - Verstecken von Metadaten (Shadow Stack):
      - Neuer zweiter Stack neben dem Hauptstack
      - Dort wird ein Duplikat der RET und BP gespeichert
      - Vergleich am Ende der Funktion

        - Inkonsistenz bricht das Programm ab
    - Access Control von Speichersegmenten
    - Randomisierung des Speicherlayouts

- Canaries:
    - Wächterwert in jedem Stackframe
    - Vor RET und BP
    - Vergleich am Ende der Funktion
    - Inkonsistenz bricht Programm ab

- Non-Executable Memory:
    - Access Control für Speichersegmente
    - Unterscheiden zwischen writeable (W) und executable (X)
    - Stack, Heap und Statische Daten sind W
    - Nur der Source Code ist X
    - Overflow ist weiterhin möglich, ABER der Angreifer kann keinen bösartigen Code mehr ausführen

- Randomization:
    - Randomisiere das Speicherlayout
    - ASLR
    - Overflow möglich, ABER Adressen sind nicht vorhersehbar

Malicious Software:

- Häufig auch Malware genannt
- Software mit bösartiger Funktion
- Breites Spektrum an Malware mit verschiedenen Typen
- Keine präzise und einheitliche Taxonomie
-

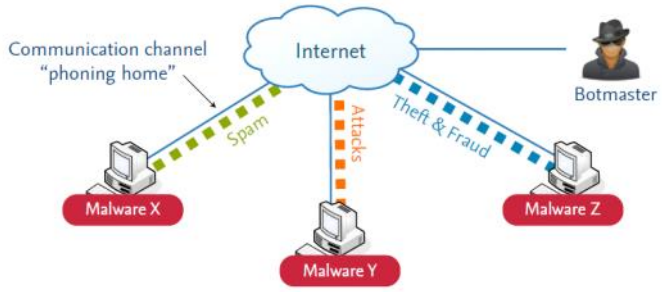
Self-repliction	Automatische Replikation und Ausbreitung
Parasitism	Injection von Malware in anderen Code
Illicit communication	(Illicit=Unerlaubte) Kommunikation mit dem Angreifer

nicht alleine lebendsfähig

Malware Communication

- **Remote control of compromised computer systems (botnet)**
    - Remote conducting of malicious activities, e.g. attacks
    - Gathering and theft of personal data, e.g. credit cards

-



- Arten:
    - Backdoors:
        - Bösartiger Code für unsichtbaren Zugriff auf Ressourcen
        - Umgehung von Authentifizierungs- und Zugangskontrollmechanismen evasion
        -

Replication: possible Parasitism: no	Communication: yes Appearance: ~1960

Script kiddie tools : Netbus, Sub7, Back Orifice
Ken Thompson Reflection on Trusting Trust

    - Logic Bombs:
        - Bösartiger Code wird an einem bestimmten Event/bestimmten Zeitpunkt ausgeführt
        - Automatischer Schaden und Sabotage
        -

Replication: possible Parasitism: no	Communication: no Appearance: ~1960

Logic bomb of R. Duronio took down 2,000 UBS servers
Michelangelo virus: Wiping of disk sectors on 6th March

    - Trojan Horses:
        - bösartiger Code, der gutartige Funktionen imitiert
        - Zusammensetzung aus gutartigen und bösartigem Code
        - Oft mit anderen Malware-Typen kombiniert

- Replication: possible
  - Parasitism: no
  - Communication: ?!
  - Appearance: ~1970

manche müssen kommunizieren manche nicht

- Computer Viruses:
  - bösartiger Code, der anderen Code mit sich selbst infiziert
  - Virencode, der in Programme oder Dokumente eingeschleust wird
  - Beliebige Nutzlast, die sich mit Computerviren repliziert

payload : Nutzlast

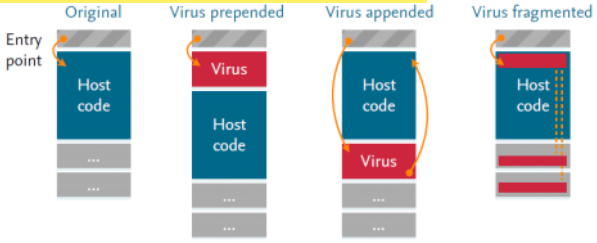
  - Replication: yes
  - Parasitism: yes
  - Communication: no
  - Appearance: ~1970
- Computer Worms:
  - bösartiger Code, der sich über das Netz selbst vervielfältigt
  - Ausbreitung durch Ausnutzung von Netzwerkschwachstellen
  - Anhängen von beliebiger Nutzlast an den Computerwurm
  - Replication: yes
  - Parasitism: no
  - Communication: yes
  - Appearance: ~1980
- Spyware & Stealers:
  - Bösartiger Code, welcher sensible Daten ausspäht

harvest

  - Funktionen für Key logging und form grabbing
  - Weite Reichweite von Adware bis Crimeware
  - Bsp: Bundestrojaner/Staatstrojaner
  - Replication: no
  - Parasitism: no
  - Communication: yes
  - Appearance: ~1990
- Bots:
  - bösartiger Code, der an einem Bot-Netz (Botnet) teilnimmt
  - Fernsteuerung eines infizierten Systems (Zombie)
  - In der Regel in Kombination mit anderen Malware-Typen
  - Replication: possible
  - Parasitism: no
  - Communication: yes
  - Appearance: ~2000

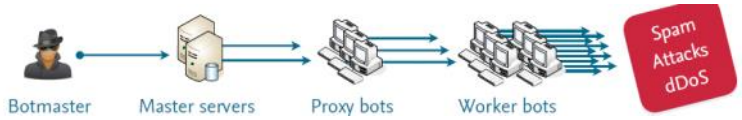
- Mechanismen:
  - File Infection:
    - Injection des bösartigen Codes in binary oder source code

entry point : pointer

    - Appending, prepending oder mixing mit vorhandener code base
    - Umleitung des Kontrollflusses vom Einstiegspunkt
    - 

  - Network Propagation:
    - Self-replication durch Netzwerkangriffe
      - Scannen und Untersuchen auf verletzliche Hosts
      - Automatische Ausnutzung von Schwachstellen und direkte Übertragung
      - Bsp: Code Red Worm (IIS), Blaster Worm (RPC)
    - Self-replication durch reguläre Kommunikation:
      - Sammeln lokaler Adressdaten (Bsp: Emails)
      - Automatischer Versand von Werbung für bösartige Inhalte
      - Bsp: Melissa Worm (Word), Samy Worm (XSS)
    - Meistens exponentielles Wachstum der Infektionen
  - Drive-by Downloads:
    - Ausnutzen von Schwachstellen im Web-Browser und derer Erweiterungen
    - JavaScript/Flash als Rahmen für die Ausnutzung
    - Automatischer Download und Infektion mit der bösartigen Software
    - Wo macht man das?
      - Injection in einer populären Webseite
      - Weiterleitung zur bösartigen Webseite (Bsp: durch Werbung die man anklickt)

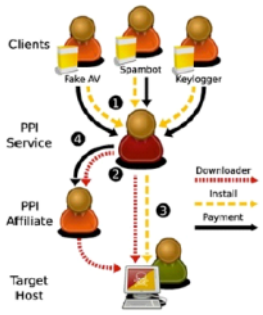
- Obfuscation:
  - Verschleierung und Verschlüsselung des bösartigen Codes
  - Polymorphismus: verschleierter code ist in jeder Kopie anders
  - Obfuscation nutzt Verschlüsselung und Kompression
  - Entpacken ist oft erst zur Laufzeit möglich
  - 

- Emulator-based Obfuscation:
  - Bösartiger Code wird in eine virtuelle Sprache L übersetzt
  - Sprache L wird für jede Kopie zufällig generiert
  - Emulation der Sprache L zur Laufzeit
  - Statische und dynamische deobfuscation schwer
    - Originaler code nur teilweise im Speicher sichtbar
    - Dynamische Analysen nicht trivial
- Cloaking & Evasion:
  - "Tarnen und Ausweichen"
  - Vor Erkennungstechniken und Analysen ausweichen
  - Tarnen der Kommunikation
  - Verstecken von Dateien und Prozessen im System
  - Anwendung von Anti-Debug Techniken
  - Checkt, ob ein virtuelles Environment eingeschaltet ist
- Botnets:
  - Netzwerk von kompromittierten Systemen
  - Bösartiges distributed system mit großer Skalierung
  - Command-and-Control Kommunikation (C&C)
  - Interne Proxys und Routing
  - Unterschiedliche Organisationsarten:
    - Hierarchisch (Zentral)
    - Peer-to-peer (dezentral)
  -
  - 

- Spam Distribution:
  - Spam Nachrichten vom Botnet
  - Übertragt Empfänger und Templates zu den Bots
  - Riesige Mengen an Spamnachrichten können verschickt werden
  - Viele Gegenmaßnahmen sind unwirksam, z. B. IP-Blacklists
- Strukturierung:

Pay-per-Install

- **Next level of malware business**
  - Outsourcing of malware infection and distribution
- **Model with different roles**
  - Providers of malicious software
  - Pay-per-install (PPI) services
  - Additional PPI affiliates
- Tracking and mitigation of such organized crime difficult



Intrusion and Malware Detection:

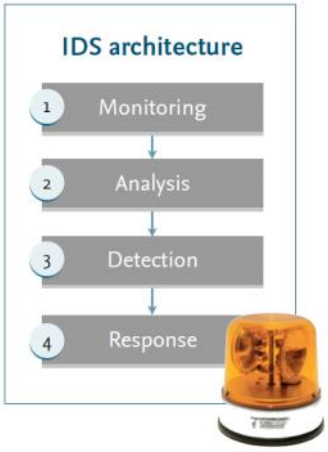
Attacke	Der Versuch ein oder mehrere Sicherheitsziele zu verletzen

Intrusion	Erfolgreiche Attacke (mehr oder weniger)
Malware	Bösartige Software, welche in einer Attacke genutzt wird

- Intrusion Detection (Einbruchserkennung):
  - Erkennung von Intrusions in Daten
  - Bsp: SNORT

- Malware Detection:
  - Erkennung von Malware in Daten
  - Bsp: Virus Scanner Clam AV

- Malware Detection = Intrusion Detection?
  - In der Vergangenheit unterschiedlich, heute teilweise gleich



- Monitoring of data
  e.g. program behavior or traffic

  Analysis of data
  e.g. parsing; extraction of features

  Detection of threats
  e.g. misuse patterns, anomaly detection

  Response to threats
  e.g. alert messages, blocking

- Monitoring and Auditing:

	Network-based Monitoring	Host-based Monitoring	Application-based Monitoring
Beschreibung	- Monitoring von Daten auf einem Netzwerkknoten - Live Verfolgung des Netzwerktraffics - Überprüfung von Paketheadern und Payloads	- Monitoring von Daten beim Host - Batch und on-access Überprüfung von Dateien - Auditing und Überwachung des Programmverhaltens	- Monitoring von Daten innerhalb einer Anwendung - Spezifisches Auditing von Anwendungslogik und Anwendungsstatus - Überprüfung von Transaktionen und Logs
Vorteil	Schutz ganzer Netzsegmente	Feinegradige Analyse der Aktivität des Hosts	Erkennung von anwendungsspezifischen Angriffen
Nachteil	Begrenzt durch Verschlüsselung und Datenvolumen	Erheblicher Overhead während der Laufzeit	Laufzeit-Overhead und Erweiterung der Anwendung
Beispiel	Network intrusion detection system (NIDS)	Klassische Virus Scanner, Datenintegritätsprüfer	Sicherheits-Plug-ins für Browser, Datenbank auditing

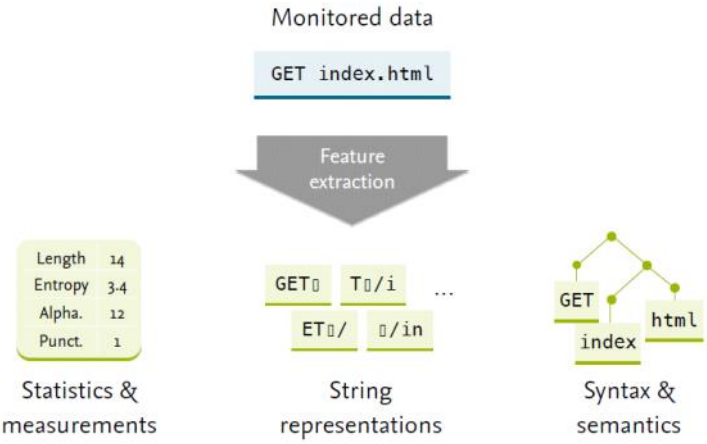
- Analysis and feature extraction:

	Network-based Analysis	Host-based Analysis
Beschreibung	- Analyse der Netzwerkdaten für Intrusion Detection - Defragmentierung und Wiederzusammensetzung des Netzwerkverkehrs - Parsing von Protokolldaten auf verschiedenen Netzwerkschichten - Analyse der Nutzdaten (--> hostbasierte Analyse)	- Analyse der Daten auf dem Host zur Intrusion Detection - Entpacken und Entschleiern von Dateiinhalten - Statische Analyse von Dateiformat und -inhalt - Dynamische Analyse (Emulation) von Code
Vorteil	Analyse der Daten vor der Verarbeitung auf dem Host	Wirksamer Schutz direkt am Angriffsziel
Nachteil	- Große Vielfalt an Protokollen und Datenformaten - Evasion: Semantische Lücke zur Verarbeitung auf dem Host	- Entdeckung von bösartigem Code nicht trivial - Einschränkung der Benutzerfreundlichkeit durch Laufzeit-Overhead

  - Feature Extraction (Merkmalsextraktion):

    statische :
    - Code der Malware untersuchen

    dynamische :
    - Code der Malware in kontrollierter Umgebung ausführen



- Detection Concepts:
  - Misuse detection:
    - Erkennung durch bekannte Missbrauchsmuster
    - Bsp: Attack signatures, behavior heuristics
      - Signature of SNORT IDS

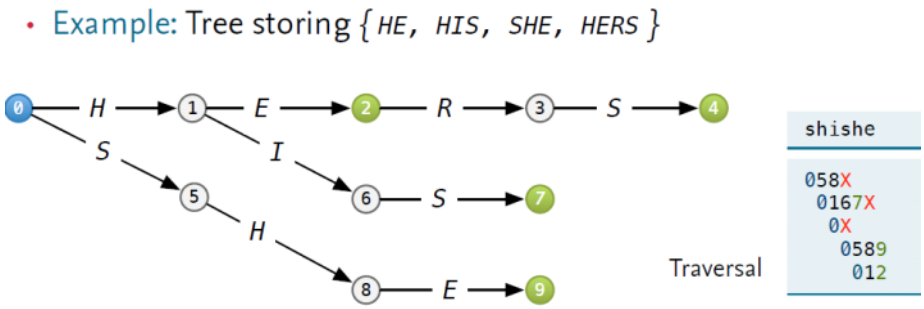
```
alert tcp $EXTERNAL_NET any -> 10.0.0.0/16 80
flow: to_server, established
content: "..%c1%9c.."
msg: "simplified signature for NIMDA worm"
```

      - Signature of BRO IDS

```
signature simple-signature {
  ip-proto == tcp
  dst-ip == 10.0.0.0/16
  dst-port == 80
  http ..%c1%9c..
  event "simplified signature for NIMDA"
}
```

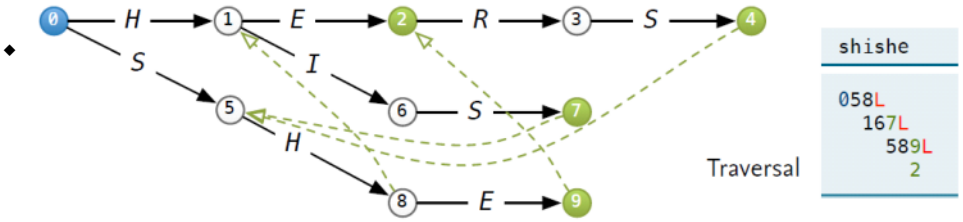
    - Beschreibung der Signaturen durch formale Sprache
      - Regular expressions, state machines, …
    - Vorteil:
      - Effektive und effiziente Erkennung von bekannten Angriffen
    - Nachteil:
      - Ineffektiv gegen unbekannte Angriffe

    - Keyword Tree:
      - Verwaltung der Bekannten Signaturen in einer effizienten Datenstruktur
      - Keyword Tree:
        - Struktur um Strings zu speichern und zu matchen
      - Signaturen werden als Pfad von der Wurzel bis zum markierten Knoten gespeichert
      - Bsp:



      - Problem:
        - Nach Mismatch muss wieder beim Wurzelknoten gesucht werden
      - Besser:
        - Aho-Corasick Algorithmus
        - Keyword Tree mit Failure Links

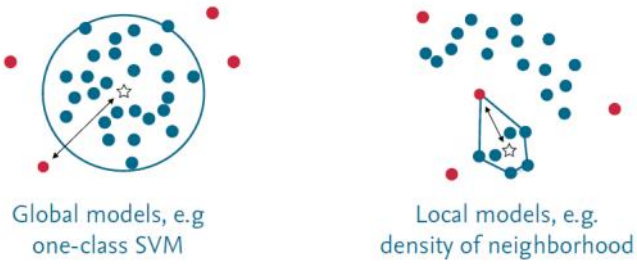
- For every node: Link longest proper suffix on path to matching prefix (if any)



- Anomaly detection:
  - Erkennung durch ein Modell der Normalität
  - --> Alles was von dieser Normalität abweicht ist eine Anomalie, also potentiell bösartig
  - Normalität ausgedrückt durch ein erlerntes Modell (Machine Learning)
  - Modelle auf der Grundlage von Spezifikationen, Statistiken, Wahrscheinlichkeiten, …
  - Vorteil:
    - Erkennung auch von unbekannten und neuartigen Angriffen
  - Nachteil:
    - Inhärente semantische Lücke: anomal ≠ bösartig

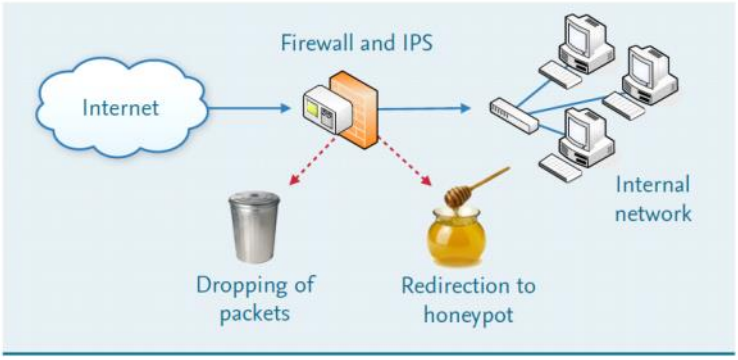
Geometric modeling of normality in feature space

- Normality described by geometric object, e.g. sphere
- Explicit or implicit compensation of "dirty" training data
- Examples: one-class SVM, density estimation, …

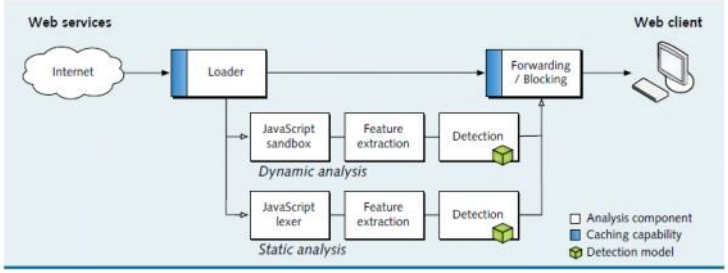


- Response and wrap-up:
  - Response:
    - Verschiedene Reaktionsstrategien auf erkannte Bedrohungen
      - Versenden oder Protokollieren von Warnmeldungen
      - Blockieren von Kommunikation und Programmen
      - Quarantäne von infizierten Dateien
    - Manchmal sogar Beseitigung möglich
      - Entfernung von bösartigem Code aus Dateien und Paketen
      - Wiederherstellung des Systemzustands vor dem Angriff (z. B. Snapshot)
    - Reaktion potenziell anfällig für andere Angriffe
      - Erneute Spoofing- und Denial-of-Service-Angriffe
    - Beispiele:
      - Intrusion Prevention System (IPS):

- Regular IDS combined with firewall mechanism
- Packets matching signatures dropped or redirected

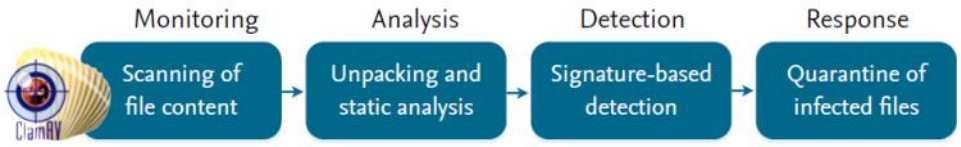


- Cujo Proxy:

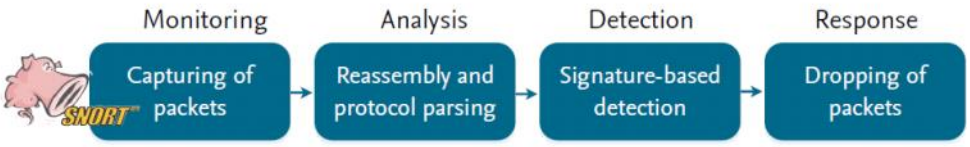


- **Web proxy capable of blocking drive-by-download attacks**
  - On-the-fly inspection of JavaScript code base
  - Detection of attack patterns using machine learning
  - Blocking of web pages containing attacks

- Alles zusammen:
  - **A classic virus scanner**

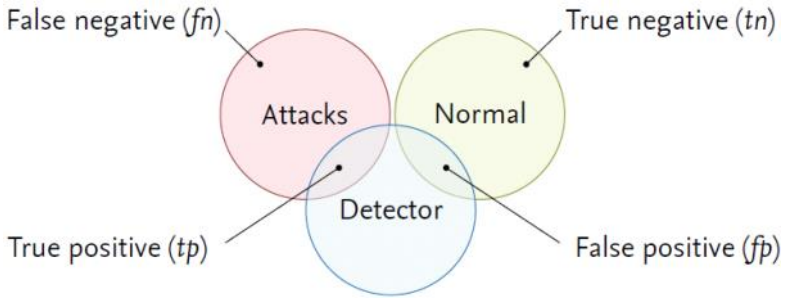


  - **A classic network intrusion detection system**



  - Note: Separation of processing steps often not clear

  - Detection Perfomrance:
    - Leistung der Erkennungstechniken
    - Zwei Arten von Fehlern: falsch-positive und falsch-negative
    - Oft ist eine Art wichtiger als die andere

- 

- Receiver Operating Characteristic (ROC) Curves:
  - Visualisierungstechnik aus dem Bereich der Signalverarbeitung
  - Erkennung variiert durch Schwellenwert (Arbeitspunkt)

Evasion and Future;

- Einige Umgehungsmöglichkeiten:

Red herring	DDOS mit gefälschten Angriffen - System wird überflutet mit möglichen Angriffen
Mimicry Attacken	Anpassung der Angriffe an normale Daten, weil Muster bekannt sind
Poisoning	Verunreinigung von Lerndaten