Introduction to Security

Zusammenfassung für das Mitdterm Exam

ri_ba, AllesBeimAlten

MSWORDFORLIFE

Kapitel 1: Übersicht (Overview)

Definition: Ist Schutz der einem automatisierten Computer-System geboten wird, um die geltenden Ziele zur Bewahrung von Integrität, Geheimhaltung und Verfügbarkeit der informationssystembezogenen Ressourcen zu erfüllen.

(Wie erwähnt) Die Ziele sind …

- Geheimhaltung: Bestimmte Daten müssen geheim gehalten werden. (Auch schon deren bloße Existenz)
- Integrität: Zugriff nur für autorisierte Personen. (Lesen, ändern, schreiben von Daten)
- Verfügbarkeit: Für den Benutzer autorisierte „Teile“ können von diesem verlässlich und „frei“ benutzt werden.

… von Daten und Services.

Die Herausforderungen bei Computer Security sind:

- Mögliche Probleme sind selten simpel und intuitiv lösbar
- „involves algorithms and secret Info“
- Die Realisierung des Schutzsystems erfolgt meist erst nach der Entwicklung des eigentlichen Einsatzsystems
- Bei der Entwicklung von Schutzsystem müssen auch immer potentielle „Gegenangriffsmöglichkeiten“ bedacht werden
- Zumeist ein geistiger Wettstreit zwischen Admin und Angreifer
- Ist ein Schutzsystem im Einsatz erfordert es ständige Beobachtung und Wartung
- Viele Nutzergruppen ziehen die meiste Zeit keinen Vorteil aus Sicherheitsmechanismen, erst dann wenn diese „aktiv zum Einsatz“ (security failure?) kommen
- Werden oft als unangenehmes (jedoch notwendiges) Hindernis zum „Einsatzsystem“ betrachtet

<table>
<tr><th colspan="2">Angriffe auf Schwachstellen und Gegenmaßnahmen</th></tr>
<tr><td>Schwachstellen können ausgenutzt werden um das System …
zu korrumpieren (Verlust an Integrität)
undicht zu machen (Verlust an Geheimhaltung)
zu zerstören / unverfügbar zu machen (Verlust an Verfügbarkeit)
Angriffe können von Innen bzw. Außenstehenden und sowohl aktiv (Veränderung des Systems) als auch passiv („nur“ Ausnutzung von Informationen) geschehen.</td><td>Gegenmaßnahmen sollen Attacken …
vermeiden,
aufspüren …
… und dann mögliche Schäden beheben.
Gegenmaßnahmen versuchen Risiko bei gegeben Einschränkungen zu minimieren. Jedoch ist kein System perfekt und es bleibt immer eine bestimmte Restverwundbarkeit. Es können durch (besonders bei schlecht geplanten) Schutzmaßnahmen sogar völlig neue Schwachstellen auftreten.</td></tr>
</table>

Weitere Definitionen:

- Adversary oder Threat Agent: Jemand er das System angreift oder gefährdet
- Threat: Ein Potential (eine potentielle Stelle/Umstand) um die Sicherheit des Systems zu gefährden
- System Resource (Asset): Daten; Ein (nützlicher, gutartiger) Service eines Systems; Systemressourcen; …
- Security Policy: dt. Sicherheitsrichtlinie, spezifiziert Regeln und Routinen für Schutzmaßnahmen um kritische Systemressourcen zu schützen.
- Risk: Risiko beschreibt die potentielle Wahrscheinlichkeit, dass ein bestimmter Threat eine Schwachstelle ausnutzt und den System Schaden zufügt

Konsequenzen von „Threats“ sind …

- unautorisierte Offenlegung (Exposure: Daten werden direkt für eine unautorisierte Entität offengelegt, Intrusion: Eine unautorisierte Entität umgeht die systemeigenen Sicherheitsmaßnahmen)
    - **unauthorized disclosure (Angriff auf Recht auf Geheimhaltung)**
- Betrug (Falsification: eine autorisierte Entität erhält falsche Informationen, Masquerade: Eine unautorisierte Entität gibt sich als eine Autorisierte aus)
    - **deception (Angriff auf Integrität)**
- Unterbrechung (Incapacitation: Verhindern oder Unterbechen von Systemoperationen durch Beschädigung einer Komponente, Corruption: Bösartige Beeinflussung von Systemressourcen)
    - **disruption (Angriff auf Verfügbarkeit und Integrität)**
- Machtergreifung (Misappropriation: Bösartige Übernahme und Kontrolle des Systems und deren Ressourcen, Misuse: Bringt das System dazu Funktionen oder Dienste zu benutzen die der System eigenen Sicherheit schaden.)
    - **usurpation (Angriff auf Integrität)**

Die Assets eines Computersystems können auf die fünf Bereiche: Hardware, Software, Daten, Kommunikations-Linien und Netzwerke aufgeteilt werden.
„Anwendungsbereiche (Scope):“ Sowohl die Speicherung **(file security)** als auch die Übertragung **(network security)** (sensitiver) Daten muss sicher erfolgen. Zugriffe auf Daten **(file protection)** und Computer **(user authentication)** müssen kontrolliert geschehen.

Netzwerk Angriffe ..

- … sind passiv … (Abhören): (unautorisierte) Veröffentlichung von Inhalten, Traffic-Analysen …
    - Schwer aufzuspüren, daher sollten sie vermieden werden
- .. oder aktiv. (Modifizieren bzw. fälschen von Daten): Maskierung, DOS, Modifkiation …
    - Schwer zu vermeiden, daher sollten sie aufgespürt werden

Security-Kassifizierung (Taxonomy-Bild):
Attackers → nutzen Tool bzw. Exploit → von / auf Systemschwachstelle → dies führt zu einer Aktion → bei einem Target(Ziel) → liefert ein unautorisiertes Ergebnis → das zur Erfüllung des Ziels (der Grund warum man überhaupt das System angreift) dient

X-800 Sicherheitsarchitektur (Netzwerk):

- … Architektur für das OSI-Referenzmodell
- Systematischer Weg um Sicherheitsansprüche zu definieren und Möglichkeiten zur Befriedigung dieser zu charakterisieren
- Definieren: Sicherheits-Angriffe (Gefährdung von Sicherheit), -Mechanismen (Vermeiden, Aufspüren, Wiederherstellen), -Services (Verbessert Sicherheit, Gegenangriffe)

Funktionelle Ansprüche:

- Technische Maßnahmen: (Zugriffskontrolle, Identifikation und Autorisierung, Systemkommunikation und -beschützung, …)
- Verwaltungsbezogene Kontrollen und Prozeduren: (Sicherheitsbewusstheit und Training, Wartung, …)
- Verwaltung- und Technikbezogene Maßnahmen: (Konfigurationsverwaltung, Schuldverantwortlichkeit, …)

Trends: Mit dem Lauf der Zeit ist sowohl sind sicherheitsbezogene Attacken sowohl durchdachter, und professioneller als auch leichter zu Handhaben geworden.

Kapitel 2: Verschlüsselung (Cryptographic Tools)

Verschlüsselung ist in sehr vielen Bereichen von Computer Security wichtig (Datenübertragung, sichere Speicherung, …).

Symmetrische Verschlüsselung:

Bei einer symmetrischen Verschlüsselung, wird im Gegensatz zu asymmetrischer Verschlüsselung, bei beiden Kommunikationspartner der gleiche (oder ein ähnlicher, z.B. bei IDEA) Schlüssel verwendet.

Schlüssel K, Daten X, Verschlüsselter Text Y, Verschlüsslung E[Daten, Schlüssel], Entschlüsselung D[Verschlüsselte Daten, Schlüssel]
SENDER(X) → *verschlüsselt mit Schlüssel* →VERSCHLÜSSELTER_TEXT(Y=E[X,K])→*entschlüsselt mit Schlüssel*→EMPFÄNGER(X=D[Y,K])

Angriff auf Symmetrische Verschlüsselung:

- Brute-Force Angriff (Alle möglichen Schlüssel auf Daten ausprobieren bis „sinnvolle“ Daten rauskommen) (Exhaustive Key-Search)
- Rekonstruktion durch Analyse (basierend auf Natur des Algorithmus und Daten / Text - Charakteristika, Nutzen bekannter „Purtext“ (Purdaten) / Cyphertext-Paare, (möglicherweise bei schlechteren Algorithmen) ausnutzen bestimmte Eigenarten des Algorithmus um Schlüssel bzw. Nutzdaten zu rekonstruieren

Symmetrische Algorithmen:

- **D**ata-ES und Triple-DES
    - DES ist der am meisten verbreitete Verschlüsselungsalgorithmus
        - 64Bit Text und 56Bit Schlüssel um 64Bit verschlüsselten Text herzustellen, Es gibt Bedenken über den Algorithmus und der (relativ kurzen) länge des 56Bit-Schlüssels (nicht ausreichend sicher)
    - Triple-DES wendet DES-Algorithmus einfach 3fach an
        - Verwendet zwei oder drei einzigartige Schlüssel
        - Wesentlich sicherer, aber auch wesentlich langsamer
- **A**dvanced-ES
    - 64Bit Text zu 64 Bit verschlüsselten Text mit 128, 192 oder 256Bit Schlüssel.
    - NIST suchte 1997 neuen Algorithmus da DES als nichtmehr sicher genug galt.
    - 2001 kam AES.
    - AES ist der shit!!

Blockverschlüsselung/Stromverschlüsselung (Block/Streamcypher):

- Wie bei DES/AES, Daten werden in Blöcke von zerlegt und verschlüsselt. ECB = Eletronic Code Book: Daten werden jeweils zu einem Block von größe b (meist 64/128Bit) zerlegt, verschlüsselt, übertragen und beim Empfänger wieder entschlüsseln. Hierbei kommt die ganze Zeit ein und derselbe symmetrische Schlüssel zum Einsatz
- Jedes einzelne Bit, dass übertragen wird, wird mit mit einem Wert aus einem Pseudo-Zufallszahlengenerator (key stream generator) XOR-Verknüpft und somit verschlüsselt. Mit gescheitem Generator, kann so etwas ähnlich gut wie eine Block Verschlüsselung sein. Vorteil: Geschwindigkeit.

Nachrichten-Authentifizierung (Message Authentication):

Nun haben wir bereits den Schutz von passiven Angriffen behandelt. (Abhören, etc.) Die Frage ist nun wie Schütz man sich vor aktiven Angriffen, ins besondere Falsification und Modification.

Message Authentication kann (und soll) nun mehrere Dinge sicherstellen:
Verfizierung ob Nachricht tatsächlich vom gedachte Absender stammt und ob dessen Inhalt unmodifziert übertragen wurden. Hierbei wäre als natürlich auch Interessant zu wissen ob die Nachrichten auch tatsächlich „fließend“ zwischen den Kommunikationspartnern übermittelt werden.

- Eine Methode dazu wäre eh einfach Verschlüsselung. Hierbei kann man relativ sicher sein, das der korrekte Absender, der einzige ist der Nachrichten korrekt ver- und entschlüsseln kann. (Sowohl Symmetrisch, Also auch asymmetrisch (public Key))
- Wenn die Nachrichten noch Fehler-Erkennung und Laufnummern beinhalten kann auch noch zusätzlich gesichert werden, dass es zu keiner nachträglichen Modifikation kam.
- Zeitstempel könnten außerdem aufzeigen, ob eine Nachricht (im Vergleich zu anderen) ungewöhnlich lange zur Übermittlung benötigt hat.

- Es gibt noch diverse andere Verfahren die nicht auf Verschlüsselung beruhen, bei all diesen wird ein Autentifzierungs-Tag generiert und an die Nachricht angehängt. Der Text kann dabei unverschlüsselt bleiben.
    - Message Authentication Codes: hierbei Teilen sich zwei Partner einen (geheimen) Schüssel K mit Hilfe dessen ein Message Authentication Code MAC aus einer Nachricht M generiert und an die Nachricht angehängt werden kann. MAC = generateKey(K,M). Sendet Partner A an B eine bestimmte Nachricht, und kennen nur beide Partner den Schlüssel B. So kann B bei erneuter Überprüfung des passenden MAC's festellten, dass die Nachricht tatsächlich von A kommt und außerdem nicht verändert wurde. Kann mit „DES" realsiert werden. (Letzte Bits von verschlüsselten Text können auch als MAC dienen, besonders interessant bei asymmetrischen Crypts → digitale Signatur)

Funktionen die Werte wie den MAC generieren heißen Hashfunktionen. An sie gibt es bestimmte Forderungen:

- 1-5 Schwach:
    - 1. Kann auf jede Länge von Daten angewandt werden
    - 2. Der von der Hashfunktion berechnete Wert hat eine fixe Länge
    - 3. Der von der Hashfunktion berechnet Wert soll relativ einfach (effizient) zu berechnen sein
    - 4. Ein-Wegs-Eingenschaft: Aus Daten ist schnell ein Hashwert generiert. Aus den Hashwert die ursprünglichen Daten zu generieren soll aber quasi unmöglich sein.
    - 5. Es soll außerdem (unmöglich) schwer sein zu einem gegeben Hashwert H(x) einen zweiten Hashwert H(y) zu finden, sodass H(x) = H(y), wobei x != y (weak collision resistance)
- 6 Stark:
    - 6. Es soll quasi unmöglich sein ein paar (x,y) zu finden, wobei x != y, sodass gilt H(x) = H(y)(strong collision resistance

Unterschied 5. und 6.: Bei gegeben Hashwert, zweiten gleichen Wert (erster Fix) finden. (5) Wertepaar finden, sodass die berechneten Hashwerte gleich sind. (beide noch Variabel) (6)

Angriffsflächen:

- Brute-force
    - Einfach durchprobieren, lange Hashwerte ist das einzige was davor gut schützt.
- Crypto-Analysis
    - Schwachstellen Auffinden im Algorithmus der die Hashwerte benutzt und ausnutzen

SHA-1 ist heute zu Tage meist genutzt. Liefert einen 160-Bit langen Hashwert (relativ sicher gegen Brut-Force) und verfügt über einen soliden Algorithmus. Alternativen (mit erhöhtem Schutz) SHA-256, SHA-384, SHA-512.

Public Key – Verschlüsselung:

Prinzip: Bei Übertragung von B zu A nutzt Sender B A's öffentlichen Schlüssel zur Verschlüsselung der zu übermittelnden Daten. Die originalen Daten lassen sich nur noch mit A's privaten Schlüssel „wiederherstellen". Kann aber auch genau umgekehrt realisier t werden, sodass B zur Übermittelung an A seinen privaten Schlüssel benutzt, wobei A mit B's öffentlichen Schlüssel die Nachricht „aufsperrt." Hierbei ist zu beachten, dass jeder der B's öffentlichen Schlüssel kennt in der Lage ist die übermittelte Nachricht zu lesen.

Öffentliche-Schlüssel O, Private Schlüssel P, zu verschlüsselnde Daten bzw. Text T und Verschlüsslungsalgorithmus E[Daten, Schlüssel], Entschlüsselung D[Verschlüsselte Daten, Schlüssel]

Sender B und Empfänger A

SENDER(X) → *verschlüsselt mit Schlüssel* O_a →VERSCHLÜSSELTER_TEXT(Y=E[X,O_a])→*entschlüsselt mit Schlüssel* P_a →EMPFÄNGER(X=D[Y,O_a])

Statt ganze Nachricht zu verschlüsseln können auch Authenticator benutzt werden. (Verschlüsselung einer Funktion die benötigt wird um tatsächliches Dokument zu lesen, z.b SHA-1-Funktion)

Anforderungen an **Public-Key Verschlüsselung**:

1. Es soll leicht sein neue Privat/Öffentlich-Schlüsselpaare zu erstellen
2. Es soll leicht sein für einen Besitzer eines Public-Keys Nachrichten mit diesem zu VERschlüsseln
3. Es soll leicht sein für einen Besitzer eines Private-Keys Nachrichten mit diesem zu ENTschlüsseln.
4. Es soll schwer sein vom Public auf den Private-Key schließen zu können
5. Es soll schwer sein ohne Kenntnis der Keys die Nachrichten zu entschlüsseln
6. Nützlich wenn beide Schlüssel sowohl für Verschlüsselung als auch Entschlüsselung benutzt werden können.

Bekannte Algorithmen:

- RSA (Blockverschlüsselung, am weitesten verbreitet, nur wirklich sicher falls Schlüsselgröße mindesten 1024Bit beträgt)
- Diffie-Hellman key exchange Algorithmus
- Digital Signature Standard
- ECC (ähnlich zu RSA, alledings neuer mit kleinen Schlüssel, noch nicht so akzeptiert wie RSA)

Ein Problem bei Public Key – Verschlüsselung ist die Ungewissheit ob man im Besitz eines autentischen Public Key's ist. Eine Lösung hierzu sind **Public Key Certificates**, wobei ein Public Key außerdem zusätzlich über eine Benutzer-ID besitzt die von einer bekannten Zertifikat-Organisation signiert wurde. (Aus Public + UserID wird ein Hashwert berechnet, der mit dem private Key der Zertifikat-Organisation verschlüsselt wird, dieser Block wird an der Public-Key/UserID „Packung" angehängt").

Digitale Signatur ist hierbei auch ein Stichwort.

Public – Key Verschlüsselung kann auch benutzt werden zur Erstellung „digitaler Briefumschläge“ und damit auch zur Übertragung von symmetrischen Schlüsseln. Kann somit zur „initialisierung“ einer symmetrisch Verschlüsselten Konversation dienen.

Random Numbers und Anhang

Bei Verschlüsselung sind Zufallszahlen sehr wichtig. Sowohl zur direkten Verschlüsselung (zb. Bei Stromverschlüsselung) als auch bei der Generierung von Schlüsseln selbst. Zufallszahlen sind nur „ECHTE ZUFALLSZAHLEN“, falls ihre Generierung völlig unvorhersehbar ist und somit auch niemals von einem Vorgänger auf einem Nachfolger geschlossen werden darf. Echte Zufallszahlen benutzen zur Bestimmung einer Folge eine nichtdeterministische Quelle.
Jedoch werden nicht immer „ECHTE“ Zufallszahlen benutzt, sondern öfter auch Pseudo-Zufallszahlen. Pseudo-Zufallszahlen wirken zufällig, werden allerdings von einem Algorithmus generiert. Hierbei ist es dann natürlich wichtig, dass trotzdem sichergestellt wird, dass Zahlenfolgen nicht vorhergesagt werden können.

Bei Datenübertragung wird Verschlüsselung sehr ernst genommen, bei Speicherung noch nicht so, obwohl diese relativ leicht kopiert und sogar wiederhergestellt werden kann.
Möglichkeiten dieses doch zu tun:

- Back-end appliance: Gerät zwischen Storage und Server verschlüsselt Daten
- Libary-based tape encryption: Bandspeicher kann verschlüsselt werden indem man einen speziellen-Co-Prozessor im Schreibe/Lese-Apparat verwendet, der die Daten mit einen nicht auslesbaren Schlüssel verschlüsselt. Daten können mit selber Apparatur jedoch wieder entschlüsselt werden.
- Background laptop and PC data encryption: Software Lösungen auf PC und laptop. (Truecrypt, etc.)

Kapitel 3: Benutzerauthentifizierung (User Authentication)

Fundamental im Bereich Computer Security. „First line of defence“.

Benutzerauthentifizierung besteht aus zwei Schritten ….

1. Identifikation - Bereitstellung eines Identifiers
2. Verifizierung – Generierung einer Authentifikations-Information zur Bindung einer Entität (Person) am Identifier

… und es gibt vier Möglichkeiten sie zu realisieren …

1. Etwas, dass der Benutzer weiß (Passwort oder PIN, …)
2. Etwas, dass der Benutzer besitzt (Keycards, …)
3. Etwas, dass der Benutzer ist (Fingerabdruck, …)
4. Etwas, dass der Benutzer (auf eine spezielle Weise) tut (Stimme, Handschrift, …)

Passwort-Schutz: Am meisten eingesetzt. (Username/Passwort) Passwort identifiziert Nutzer und Nutzer verfügt über gewisse Privilegien.

Verwundbarkeiten (Angriffsmöglichkeiten):

- Wörterbuch-Attacke
- Account-Spezifisches-Raten
- Populäres-Passwort-Raten
- … und weitere auch eh klare. (Workstation Hijacking, …)

Gegenmaßnahmen:

- Sicherheitsmaßnahmen die unerlaubten Zugriff auf Password-File vorbeugen
- Eindringlings-Detektions-Mechanismen
- Account-Sperrungs-Mechanismen (Sperrung nach X gescheiterten Login-versuchen)
- Richtlinien gegen die Benutzung schwacher und populärer Passwörter
- Training und Bewusstmachung von sicherer Passwortwahlen
- Automatisches Ausloggen bei Benutzerinaktivität
- encrypted network links (ja, genau?)

Zur Erhöhung der Sicherheit werden Passwörter auch öfter als Auswertung einer Hashfunktion mit einem zufälligen “Salt” in dem Password-File gespeichert. Der Salt muss hierbei auch irgendwo (damit Login mit dem Urpasswort noch funktioniert) vermerkt werden. Bei Wörterbuch-Attacken auf gehashte Passwort-Files werden oft Listen an vermuteten Passwörtern-gehasht und mit den Einträgen der Passwort-File vergleicht. Der Salt lässt die gespeicherten Werte, sozusagen „anders“ aussehen.

Implementationen:

Unix, gilt heutzutage als nicht mehr wirklich sicher:

- 8 Buchstaben langes Passwort wird in einen 56-Bit DES-Schlüssel konvertiert. Dieser wird dann mit crypt(3) (DES-basierende Hashroutine) auf einem 64er-Block an 0ern 25mal ausgeführt. Ein 12Bit Salt wird hierbei verwendet.

Andere benutzen natürlich leiwandere Salt/Hash-Kombinationen. Sehr beliebt: MD5 mit Salt bis zu 48Bit und Beschränkungen zur Passwortlänge. Das ganze liefert dann einen 128Bit (1000x iterativ gehasht) Wert, ist dabei allerding wesentlich langsamer als crypt(3). Dann gibt’s noch so Dinge wie Blowfish: Passwörter bis zu 55-Chars liefern Hashwert von 192Bit.

Angriffe darauf sind eben Wörterbuch (Mögliche Passwörter, deren Variationen mit allen Salt-Möglichkeiten durchprobieren) und Rainbow-Table-Angriffe (Ähnliches nur, dass Hashwert vor Angriff berechnet und in einem Rainbow-Table gespeichert wird, erspart Zeit).

Schlaue Sache ist natürlich direkten Zugriff auf Passwort-File nur sehr privilegierten Benutzern zu gestatten, bzw. Usernamen und Passwörter getrennt zu speichern. (Shadow Password File) 100% zuverlässiger Schutz ist dieser natürlich trotzdem nicht. OSs können exploitet werden, dumme User machen dumme BackUps von Passwörtern etc.

Verbesserungen:

- Leute trainieren und unterrichten bessere Passwörter zu Nutzen
- Reaktiver Passwort-Test (System rennt Cracker selbst um Lücken zu finden), Proaktiver-Test System testet Passwort bei Erstellung und checkt ob Qualität gut genug
    - Passwörter zumindest 8 Buchstaben lang, und soll eine Mischung aus Zahlen, Klein und Großbuchstaben haben
    - Markov-Modell: Alle unerwünschten Passwörter generieren und falls gewählt verweigern.
    - Bloom Filter: WUT?

Token Authentication = User identifiziert sich mit Objekt (Biometric ID-Card, …)
Memory Card: Magnetstreifen (z.B. Bankkarte) Nachteile/Vorteile trivial, _nur_ Speicherkarte
Smart Card: Verfügt auch über I/O Ports, (EEP)ROM (für Programme-Protokolle), RAM und Prozessor, vielleicht sogar Krypt-Co-Prozessor. Ähnlich: USB-Dongle

Biometrische Identifikation:

Identifiziert Nutzer durch physikalische Charakteristika: Stimme, Iris, Fingerabdruck, Handfläche, Retina, Gesicht

Beispielsystem: Zuerst muss Nutzer mit Merkmal, Name und PIN (o.ä.) ins System aufgenommen. Zur Authentifizierung gibt der Benutzer den PIN ein und lässt sich vom „Merkmalscanner“ (z.b. Fingerabdruck) analysieren. Das System identifiziert dann den Benutzer per Fingerabdruck und authorisiert ihn zugleich durch den PIN. Mit Hilfe des Fingerabdrucks allein kann der User sich auch einfach nur identifizieren lassen. Die „Erkennung“ folgt natürlich nicht sofort und 100% übereinstimmend. Das User wird bei der Charakterstik-Abnahme mehrmals mit dem System vergleicht und erhält sozusagen eine „Punktezahl“ für die Gleichheit. Übersteigt die einen bestimmten Wert, wird dieser authentifiziert. Gibt hierbei natürlich Charakterstika die besser als „eindeutig“ gematcht und andere die schlechte gematcht werden. (Dazu kann man auch ne Kurve plotten)

Netzwerk Identifikation:

Ident übers Netzwerk natürlich viel unangenehmer als lokal. Viele Angriffspunkte:
DDOS (Nutzer „deaktivieren“), Lauschangriff, Replay (gelernte Daten z.b. aus Lauschangriff wiederholt an Server senden), …

Dazu challenge/response (User = Client)

1. User schickt Identität
2. Server Antwortet mit „nonce“ (Zufallszahl r), das ist die „Challenge“
3. User wertet eine Funktion mit Hashwert h von Passwort P und r aus (f(r,h(P)) und schickt an Server
4. Falls das so ist wie Server sich erwartet hat, ist User authentifiziert

Schütz vor vielen Attacken (r z.b. gegen eine „Replay“-Attacke, da ja immer neugeneriert)

NUN FOLGEN IM KAPITEL 2 FALLBSPS DIE WÜRDE ICH EMPFEHLEN DURCHZULESEN!

Kapitel 4: Zugangskontrolle (Accesscontrol)

Das Kapitel ist wiedermals ein sehr wichtiger Punkt in Bezug auf Computer Security.
Es geht hierbei darum unauthorisierten Benutzern den Zugriff auf Ressourcen zu veweigern und legitimen Benutzern nur autorisierten Zugriff zu gestatten.

Wichtige Punkte hierzu:

- Authentifizierung – Sicher stellen der Identität des Benutzers
- Autorisierung – Gewähren von Rechten über bestimmte Systemressourcen an authentifizierte Benutzer
- Kontrolle – Eine Unabhängige Überprüfung und Untersuchung des Systemverlaufs und von Aktivitäten auf Einhaltung der geltenden Richtlinien (in Bezug auf Zugangskontrolle). Sowohl um Rechtsbrüche aufzuspüren als auch um angebrachte Änderungen in Systemverhalten und Richtlinien (in Bezug auf Zugangskontrolle)aufzuzeigen.

User müssen also vom System authentifiziert werden und bei gewünschten Ressourcenzugriffen in einer vom System verwalteten Datenbank einsehen ob der Benutzer dazu berechtigt ist und im gegebenen Fall die Ressource ausliefern. All das wird eben von einem Unabhängig Medium (Prozess, wasauchimmer) beobachtet und aufgezeichnet.

Richtlinien (in 3 Gruppen aufgeteilt):

- Discretionary AC (DAC): Bestimmt basierend auf Identität des Anforderers (der Ressource) und der Zugriffsregeln was der Anforderer tun darf und was nicht. (Entität darf Anderen Rechte geben)
- Mandatory AC (MAC): Bestimmt basierend auf Security-Labels (indizieren wie sensibel bzw. kritisch eine Ressource ist) und Security-Clearances (indizieren welche Entitäten sind berechtigt auf welche Ressourcen zuzugreifen) (Entität darf Anderen keine Rechte geben)
- Roled-based access control (RBAC): Basierend auf der Rolle die der Nutzer im System hat werden ihm bestimmte Rechte gewährt.

DAC ist traditionell im Computer Bereich. MAC ist eher so militärig. Und RBAC wird immer populärer.

Anforderungen an Zugriffssysteme:

- Erwartet bereits einen vertrauenswürdigen Input (Entität schon authentifiziert) (Muss von Apparatur schon davor sichergestellt werden)
- Fein-körnige und grobe Zugriffspezifikationen (Feiner und wohl unterschiedener Zugriff für bestimmte Nutzergruppen, von Admin aber auch grober für bestimmte Klassen anlegbar)
- Least-privilege: Jeder sollte gerade so viele Rechte kriegen wie er braucht
- Offene und Geschlossen Richtlinien: White und Blacklists ;) (Zugriff nur für, Zugriff für alle außer)
- Kombination von Richtlinien, Konflikt Lösung: Auf bestimmte Ressourcen könnten mehrere Richtlinien Kombiniert Angewandt werden, und hierfür muss es auch Möglichkeiten dadurch enstehende Konflite zu lösen
- Administrative Richtlinien: Es muss spezifiziert werden können, wer Richtlinien modifizieren, vergeben etc. darf, außerdem soll es Mechanismen geben um diese durchzuführen

Elemente bei Zugriffskontrolle:

- Subjekt – Ein Benutzer (meist Repräsentiert durch (Benutzer-)Prozess) der Zugriff auf Ressourcen hat. 3 Subjekt-Klassen: Owner (bei Dateien zurückverfolgen wer hat's erstellt/geändert/…), Gruppe, Rest (Welt)
- Objekt – Ressource auf die von Subjekt Zugegriffen werden kann (Dateien, I/O-Geräte, Mails, …)

DAC ist meist als Zugriffsmatrix repräsentiert: Eine Dimension repräsentiert den User, die andere Dimension die Objekte, meist dünn besiedelt (leere Einträge)

Zugriffskontrollstrukturen

- ACL: Für jedes Objekt werden User und deren Rechte gelistet, gibt auch default-Rechte-Eintrag,
- Capability tickets: Für jeden User werden Objekte und Zugriffsrechte gespeichert. Jeder User hat eine bestimmte Anzahl solcher Tickets der er auch vergeben und „verliehen" kann. Aber grad weil sie rumgereicht werden nicht so sicher wie ACL.

Bei Capability Tickets findet man leichter raus, worauf ein User überall zugreifen kann, bei ACL auf welches Objekt welche User zugreifen können.

Access Control Model von Lamspon, Graham und Denning (zu DAC)
Das Modell beinhält einen Satz von Subjekten, einen Satz von Objekten und einen Satz von Regeln die die Zugriffsrechte der Subjekte auf die Objekte bestimmen.
Das Modell befriedigt alle Anforderungen die zu Zugriffs-Kontrolle gegeben sind (Repräsentation des **Zugriffrecht-Status**, **Durchführung von Zugriffsrechten** und die **Autorisierung für bestimmte Subjekte bestimmte Zugriffsrechte zu ändern**)

Wird dann auf der Folie dann „umfassender" erweitert. Außerdem wird der Begriff Protection State definiert. (Satz von Informationen der den „punktuellen, temporären" Zustand der Zugriffs-Rechte, die zwischen Subjekten on Objekten existieren beschreibt.) Hierauf sollte ein Blick geworfen werden.

Der Zugriff auf Ressourcen kann auch funktionell betrachtete werden. Eine Zugriffsabfrage setzt sich sodann aus einem Subjekt S, einen Typ von Zugriff a und einem Objekt X zusammen also f(S,a,X). Das Zugriffskontroll-System musst dann determinieren ob $a \in A[S,X]$. Genaueres: Siehe Grafik auf entsprechender Folie

Protection Domains
Ein Protection Domain ist Satz an Objekten mit Zugriffsrechten auf diese Objekte.

Bezüglich der Zugriffsmatrix ist eine Zeile ein Protection Domain. Bis jetzt wurde jeder Nutzer wurde mit einer eigenen Zeile versehen. In diesem Modell hat jeder Benutzer seine eigene Protection Domain, und jeder vom Benutzer gestartete Prozess hat auch (logischerweise) auch diesselben Rechte, definiert durch eben diese Protection Domain (des Benutzers).

Ein generelleres Konzept von Protection Domains führt zu mehr flexibilität: Z.B. könnten Nutzer Prozesse nur mit Subsätzen der Rechte der eigenen Protection Domain starten. Dies ist sehr nützlich da ein Benutzer so Programme denen er weniger Vertraut, in einer „neuen" eingeschränkteren

Protection Domain starten kann. Das Bündnis zwischen Protection Domain und Prozess könnte auch dynamisch sein.

UNIX (WOZUAUCHIMMER)

Dateikonzepte

Kontrolle über Datein per INodes (Index Nodes). Ein Inode ist eine Kontrollstruktur, die wichtige Information für das Betriebssystem zu einer Datei beinhält. Mehrere Dateinamen können mit ein und derselben INode verbunden sein, jedoch gehört eine aktive INode immer zu einer Datei und jede Datei wird von genau einer INode kontrolliert. Auf der Festplatte befindet sich eine INode Tabelle die alle INode's enthält. Wird eine Datei geöffnet, wird die zugehörige INode in den Speicher geladen. In den Ordner sind zu jeder Datei Pointer auf die jeweilige INode gegeben.

Dateizugriffskontrolle

Bei UNIX-Systemen hat jeder User eine ID und gehört zumindest einer (Primär-) oder mehreren Gruppen an. Erstellt ein Nutzer eine Datei ist dieser als Besitzer eingetragen, außerdem wird seine Primärgruppe vermerkt bzw. die Gruppe des „Parentdirectorys" (setGID). Mit jeder Datei sind 12 „Protectionbits" in der dazugehören INode gespeichert, wobei jeweils 3 Bits davon die RWX-Rechte des Besitzers der Datei, dessen zugehörigen Gruppe und des „Restes" beschreiben. (insgesamt also 9Bit)

Ist eine ausführbare Datei mit setUID oder setGID versehen, so werden ihr bei der Ausführung die Rechte des Besitzer (UID) oder die Gruppe des Besitzer (GID) gegeben. So wird bei Zugriffskontroll-Entscheidungen bezüglich dieses Programms nicht die tatsächliche UID des ausführenden Benutzers, sondern eben die des eigentlichen Besitzer oder dessen Gruppe benutzt. Dieses Feature gibt den Nutzer die Möglichkeit mit dem Programm Zugriff auf Ressourcen zu kriegen, über die er sonst nicht verfügen könnte.

Das letzte der 12„Protectionbits" stellt das Sticky-Bit dar~~. Ist es gesetzt wird der Speicher des Programmes auch nach dessen Beendigung im Hauptspeicher belassen.~~ Nicht mehr: Wenn gesetzt, dann darf nur noch Besitzer Datei verschieben, umbenennen oder löschen.

Dann gibt's noch den Superuser, der ist ausgenommen von allen üblichen Zugriffskontrollrestriktionen. (Er darf halt alles) (Aufzupassen ist wenn Superuser ein Programm mit SetUID erstellt)

Acess Controll Lists in UNIX

Viele moderne UNIX System benutzen ACLs. Hierbei kann eine ***beliebige*** Anzahl von Nutzern und Gruppen mit einer Datei assoziiert werden, jede mit drei eigenen Protection-Bits (rwx). ACL ist hierbei aber optional, da es immer noch die klassische „Zugriffsverwaltung" (per INode) gibt.

FreeBSD und viele andere UNIX-System benutzen eine erweiterte ACL mit folgender Strategie:

- Die Besitzer- und andere Einträge haben dieselbe Bedeutung wie immer
- Der Gruppen-Klassen Eintrag spezifiziert in dieser Installation nur noch "Gruppenrechte". Diese Rechte repräsentierten die maximalen Rechte die (per ACL) zu dort genannten Usern und Gruppen vergeben werden können. (Ausnahme: Besitzer)
- Man kann eben noch beliebig Benutzer und Gruppen zur Datei „hinzu"assoziieren.

Wenn ein Prozess nur ein Dateisystem-Objekt haben will wird zuerst eine möglichst gut passende ACL gesucht, ACL Einträge werden dann in einer bestimmten Ordnung durchsucht (Besitzer, named

users, …) Nur ein einziger Eintrag bestimmt den Zugriff. Dann wird noch überprüft ob der/die passende(n) Eintra/äge (kann einer, oder mehrere sein) über genug Berechtigungen verfügt, sodass gewünschte Operation ausgeführt werden darf.

Role-based Access Control

Bei RBAC werden Rechte nach Rollen (orientiert an den „Jobs“ die sie in der Organisation verrichten) und nicht nach Individuellen Nutzern vergeben. Die Rollen werden hierbei entweder dynamisch oder statisch an Benutzervergeben. Wobei sowohl ein Benutzer viel Rollen haben kann, als auch eine Rolle viele Benutzer. Selbige Beziehung gilt auch für Objekte.

Als Matrix dargestellt gibt es bei RBAC zwei unterschiedliche Matrizen. Die erste stellt Benutzer in der einen Dimension den Rollen in der anderen Dimension gegenüber. Die zweite ist dieselbe wie die von Lamspon, Graham und Denning nur, dass die eine Dimension von Rollen statt Benutzern repräsentiert wird. Natürlich sind auch hier wieder Dinge wie Least Privileges zu beachten.

Bei RBAC-Systemen kann man nun eine Reihe an Funktionen und Services inkludieren. Hierzu definiert **[SAND 96] ein Referenzmodell**, wobei RBAC0 die Minimalfunktionalität repräsentiert. RBAC1 = RBAC0 + Rollen Hierarchien (man denke an generalisierte Rechte), RBAC2 = RBAC0 + Beschränkungen bezüglich der Konfiguration von Komponenten des RBAC-Systems. RBAC3 = RBAC2 + RBAC1.

RBAC0 hat folgende 4 Entitätstypen:

- Benutzer (Hat UserID, benutzt System)
- Rolle (Eine benannte Job-Funktion innerhalb der Organisation das dieses Computersystem steuert)
- Permissions (Eine Erlaubnis für einen bestimmten Zugriffsmodus auf ein (oder mehrere) Objekt-e)
- Session (eine Abblidung zwischen den Benutzer und den aktivierten Subset an Rollen denen er zugewiesen ist)

Das **NIST-Modell** definiert Features in 3 Kategorien …

- Administrative Funktionen: Bieten die Möglichkeit RBAC-Komponenten zu erstellen, zu löschen und zu ändern.
- Systemunterstützende Funktionen: Funktionen für Sessionmanagement und Zugriffskontroll-Entscheidungen.
- Begutachtende Funktionen: Bieten die Möglichkeit RBAC-Komponenten und Relationen abzufragen.

… und 4 Komponenten:

- Core RBAC (Benutzer, Rollen und Permissions)
- Hierarchial RBAC (Wie RBAC1)
- Static Seperation of Duty (SSD) relations
- Dynamic Seperation of Duty (DSD) relations

SSD und DSD fügen Beschränkungen (man denke an RBAC2) zum NIST-Modell hinzu. Diese äußern sich in der Form von aufgabenverteilten Beziehungen, die benutzt werden um „Conflict-of-Interest“-

Richtlinien durchzusetzen. Solche Richtlinien werden von Organisationen möglicherweise verwendet, um Benutzer davor zu hindern ein übersteigendes Level an Autorität in Relation zu ihrer Position zu erlangen.

Kapitel 5: Datenbank-Security (Database Security)

In den Folien kommt zuerst einiges Grundlegendes zu Datenbanken. Vieles kennt man schon einiges ist mehr oder weniger wichtig. Die wichtigsten Dinge hierbei:

- DBMS ist das Datenbankmanagementsystem biete Programme und Funktionen zur Verwaltung von Datenbank, mit DDL – Data Definition Language können Entwickler Datenbanken anlegen, bzw. Daten erstellen und entfern, mit DML – D Manipulation L können Daten ausgelesen und verändert werden. Alles also iwie SQL. Zugegriffen wird per Filemanager bzw. Transactionmanager. Außerdem gibt‘s zwei zusätzliche Tabellen zur Autorisierung von Benutzer (Authorization Tables) und zur Verhinderung von Konflikten, zu denen es bei simultaner Ausführung im Konflikt stehender Befehlen geben kann
- Wenn Anwendung oder Benutzer (per Webformular, wieauchimmer) auf Datenbank zugreift geschieht das immer (wenn auch im Hintergrund) per Abfragesprache. (zb. SQL)

Datenbank Zugriffskontrolle

Jedes DBMS verfügt über System zur Zugriffskontrolle. Diese gehen auch wie die Konzepte des vorherigen Kapitels davon aus, dass ein Benutzer schon erfolgreich authentifiziert wurde.
Kommerzielle DBMS bieten DAC oder Role-based AC an.
Typischerweise unterstützen DBMS-System eine Reihe an administrativen Richtlinien (Funktionen wirkt hier aber sinnvoller):

- Zentralisierte Administration (Kleine Usergruppe (Admins) können Rechte vergeben und nehmen)
- Besitzer-basierte Administration (Der Besitzer einer Tabelle kann Zugriffsrechte für diese vergeben und nehmen)
- Dezentralisierte Administration (Wie Besitzer, nur das Besitzer seine Rechte weitergeben kann)

Die bei der Rechtevergabe werden natürlich die einzelnen Aktionstypen (select, insert) unterschieden. Die Rechte können generell, aber auch Tabellen, Spalten/Zeilen und sogar Inhaltsbezogen vergeben werden.
SQL kennt hierfür die zwei Befehle GRANT und REVOKE.

```
GRANT { privileges | role } [ON table] TO { user | role | PUBLIC }
[IDENTIFIED BY password] [WITH GRANT OPTION]
             e.g. GRANT SELECT ON ANY TABLE TO ricflair
REVOKE { privileges | role } [ON table] FROM { user | role | PUBLIC
}
             e.g. REVOKE SELECT ON ANY TABLE FROM ricflair
```

GRANT und REVOKE kann auch kaskadiert werden. Hierbei kann es zu Komplikationen (naja eher komplizierten Abläufen) kommen. Ein Beispiel hierzu ist mit Erklärung und Bild auf den Folien zu finden.

RBAC funktioniert auch in DBMS-Systemen, und das sehr gut: Es steigert die Sicherheit und mindert die Belastung des Admins.
In einer DAC-Umgebung kann man grob folgende Datenbank-Benutzer definieren:

- Endbenutzer (Benutzt auf der Datenbank aufsetzende Applikationen)
- Benutzer (Der Datenbank-Objekte besitzt und möglicherweise auch Rechte vergibt)
- Administrator (Generelle-Rechte vergabe, (allgemein) Systembetreeung)

Eine RBAC-Datenbank Einrichtung muss Rollen erstellen und löschen, und diesen Rollen Zugriffsrechte geben und nehmen können. Beispiel hierzu: MSQLServer, hat 3 Rollen-Typen: Server-Rolle, Datenbank-Rollen und Benutzer-definierte Rollen

Inference (Rückschluss/Schlussfolgerung)
In der Datenbank-Security bedeutet Inference der Prozess mit Hilfe autorisierter Abfragen unautorisierte Informationen (als legitime Antwort auf die autorisierte Abfrage) zu bekommen. Dieses Problem tritt besonders dann auf wenn die Kombination einer Datenmenge ***sensibler*** ist als die Einzeldaten, oder wenn diese Kombination genutzt werden kann um auf sensiblere Daten zu schließen. Der Weg auf dem die unautorisierten Daten erlangt werden heißt „Inference Channel“. Ein gutes Beispiel mit Bild ist hierzu wieder auf den Folien gegeben.

Gegenmaßnahmen:

- Inference-Möglichkeiten können schon beim Design bemerkt und durch Strukturänderungen behoben werden
- Erkennen von „Inference-Channel“ bei Query-Auswertung, in diesem Fall wird die Query dann schlicht und einfach abgelehnt
    - Hierfür wird natürlich ein Algorithmus benötigt der Inference erkennt, Entwicklung eines solchen ist aber eine recht komplexe Sache

Statistische Datenbanken (SDBs)
=Datenbanken die einzig statistische Werte beinhalten. (counts, averages) Können rein statistisch sein, oder statistischen (mit Nutzern die nur Zugriff auf statistische Abfragen haben) und normalen Zugang haben (mit Nutzern die Zugriff zu bestimmten Teilen per RBAC, DAC oder MAC haben). Der statistische Zugriff soll hier natürlich auch so erfolgen, dass sensible Daten des einzelnen nicht auf irgendeine Weise entblößt werden. (Gehört zu Inference) Daher muss erkannt werden wenn ein Nutzer versucht mit einer Reihe an statistischen Abfragen auf einzelne Informationen zu schließen.
Die Abfragen hierbei sind nach charakteristischen Merkmalen. Zu den „Treffern“ unter den Datensätzen wird ein STATISTISCHER WERT, doch kein tatsächliches Datum zurückgeliefert. Durch geschickte Abfragen könnte man eben beginnen auf einzelne Elemente zu schließen.

Beispiel hierzu wieder auf den Folien

Schutz gegen Interference
SDB-Implementierer haben zwei distinktive Möglichkeiten sich vor Inference zu schützen:

- Eine einfache Möglichkeit besteht darin einfach Abfragen an die Datenbank zu verweigern. Bei SDBs kann man hierfür einach ein k festsetzten, sodass eine Abfrage falls diese (wenn

auch nur statistisch Verarbeitet) X > k Werte zurückliefern würde, einfach abgelehnt wird. (Query Restriction)

- Antworten auf statistische Abfragen werden immer nur approximativ beantwortet. Das kann entweder realisiert werden indem man die Daten in der SDB modifiziert, sodass kein Rückschluss auf individuelle Datensätze gemacht werden kann (data perturbation, zb. Data-Swapping und Statistik von Wahrscheinlichkeitsverteilung der Daten berechnen), oder die Datenbank generiert Antworten die nur approximativ dem statistischen Resultaten der Abfragen auf den tatsächlichen Datensätzen entsprechen. (output perturbation, z.b. nur zufällig (oder gut) gewählte Untermengen der spezifizierten Daten zur Auswertung der Statistik nehmen, oder einfach Output bisschen zufällig oder bestimmt verfälschen)

Tracker-Angriffe

Man kann Query Restriction auch überlisten, indem man statt der konkreten, verbotenen Abfrage zwei Superabfragen tätigt. Aus deren Durchschnitt man auf die gewollten Datensätze schließen kann. Beispiel Hierzu wieder auf Folien oder in Kurzform:

Abfrage von count(C) ist verboten → C = Schnitt von C1 und C2 → Abfrage count(C1) – count(C1 aber nicht in C2), RESULTAT ➔ C

Weitere Query-Resriktionen

- Erlaubten Overlap (bzw. Schnittfläche) zwischen neu verfassten und vorhergehenden Queries begrenzen (Schlecht weil: Cooperation diverser Nutzer nicht auzuschließen, behindert auch bestimmte gutartige Abfragen (Menge und deren Teilmenge), …) (Query-Set Overlap Kontrolle)
- Nicht alle Overlaps verbieten, sondern Daten in bestimmte Gruppen teilen auf die nicht gleichzeitig Abfragen getätigt werden dürfen. (Partitionierung)

Schwachstelle an allen diesen Methoden ist das man auch an der bloßen Ablehnung mehrerer (weniger) Abfragen schon Schlüsse zum tatsächlichen Datenbestand machen kann. Ein Mittel hiergegen wiederum ist es die Abfragen des Nutzers aufzuzeichnen, zu analysieren und dementsprechend zu Handeln.

Database Encryption

Datenbanken sind meist typischerweise sehr wertvolle Datenspeicher, daher werden sie (wie auch schon bereits erläutert) mit einer großen Zahl an Sicherheitsmechanismen versehen. (Firewalls, Zugriffskontrolle, …). Daher stellt es natürlich auch einen Anreiz dar die Datenbank zumindest partiell zu verschlüsseln.

- Eine Möglichkeit hierzu ist es die komplette Datenbank zu verschlüsseln, aber die Schlüssel nicht beim Datenbank-Server zu speichern. Das ist aber sehr unflexibel und uneffizient. (Für Abfrage muss Benutzer mit Schlüssel komplette Tabellen runterladen, diese bei sich entschlüsseln und schließlich die Abfrage darauf ausführen lassen.
- Eine andere schon schlauere Möglichkeit ist es jedes Feld separat, jedoch mit den gleichen Schlüssel zu verschlüsseln. Der Server hat hierbei wieder keinen Schlüssel, sodass die Daten sicher am Server liegen und der Client bei den Abfragen einfach die Felder entschlüsselt und dann die benötigten Daten selektiert. Auch das ist nicht optimal da hier nicht die original Ordnung der Werte erhalten bleibt.

- Die letzte und beste Möglichkeit ist es die Datenbank Zeilenweise (als Block) zu verschlüsseln. Die Tabellen können hierbei mit Indexen ausgestattet werden, wobei diese auch nur für einige Werte-Blöcke angelegt werden können, sodass Angreifer die Ordnung nur grob abschätzen können.

Bei Database Encryption sind vier Entitäten involviert:

- Datenbesitzer: produziert die sensitiven Daten
- Benutzer: Stellt Anfragen an das System (um selektierte Daten zu erhalten)
- Client: Front-end für Benutzer um Abfragen an die (verschlüsselte) Datenbank zu stellen
- Server: Erhält verschlüsselte Daten vom Besitzer, und macht dieser Verfügbar für Clients

Wenn man vom vorher erwähnten „System" ausgeht, so dass der Nutzer einen Schlüssel hat und der Server nicht, könnte eine Abfrage wie folgt ablaufen:

1. Benutzer will ein oder mehrere Datenabfragen, SELECTed diese z.B. per Primärschlüssel
2. Query-Prozessor beim Client verschlüsselt den Primärschlüssel und leitet ihn weiter an den Server
3. Server tätigt Abfrage mit verschlüsselten Schlüssel und liefert verschlüsselte Ergebnisse zurück
4. Query-Prozessor beim Client entschlüsselt die zurückgelieferten Ergebnisse. Resultat: Normale, nicht verschlüsselte Datensätze beim Client

Chapter 6 – Intrusion Detection

Intruders („Eindringlinge")

- können gut-/bösartig sein (benign/hostile)

Arten:

- durch Benutzer (**user trespass**): Machtmissbrauch („privilege abuse") oder „unauthorized logon" (loggt sich ein, obwohl er nicht darf? Macht für mich irgendwie keinen Sinn...)
- durch Programme (**software trespass**): Viren, Würmer, Trojaner
- verschiedene Klassen: Hochstapler („**masquerader**", man versucht unautorisiert auf einen fremden autorisierten Account zuzugreifen), Machtsmissbrauch („**misfeasor**", Autorisierung missbrauchen oder versuchen autorisiert (illegal) mehr Rechte zu erlangen), oder heimlich („**clandestine-user**", z.B. Admin, der seine Spuren wieder verwischt, kann aber auch Außenseiter sein)

Beispiele fürs Eindringen:
am besten Folie anschauen, einige Bsp: Passwörter erraten/bruteforcen, einen packet sniffer laufen lassen, anschauen oder runterladen von wichtigen Daten und Datenbanken, ... bis physikalischen Attacken wie Workstation HiJacking

Definition „Security Intrusion":
Ein oder mehrere Ereignis(se), bei denen ein Eindringling einen Versuch startet, Zugriff zum System oder einem Teil davon zu erlangen, erfolgreich oder nicht.

Definition „Intrusion Detection“:
Ein Dienst, der Systemereignisse überwacht und analysiert, um unauthorisierte Zugriffe (fast) in Echtzeit zu finden und davor zu warnen.

Hacker

- motiviert von Nervenkitzel („Thrill“) durch Erlangen von Status (die „Hacking Community“ ist eine „Meritocracy“, d.h. Der Status wird durch das Können bestimmt
- gutartige Eindringlinge eigentlich kein Problem („wollen nur ausprobieren“), aber sie verbrauchen trotzdem Ressourcen und man kann es eben vorher nie wissen
- **IDS** (Intrusion Detection Systems), **IPS** (Intrusion Prevention Systems) und **VPN**s bzw. Netzwerk-Abschottung können dagegen helfen
- führten zu Bildung von **CERT**s (Computer Emergency Response Team), die Informationen über Schwachstellen und die Handlungsmöglichkeiten in solchen Fällen sammeln und verbreiten
- Hacker finden immer neue Wege, aber oft gleiche Verhaltensmuster
- Vorgehensbsp: Ziel auswählen, Netzwerk nach erreichbaren Diensten scannen und angreifbare erkennen, Passwort bruteforcen, „remote administration tool“ installieren, Warten auf Login des Admins und sein Passwort mitschreiben, damit auf den Rest des Netzwerks zugreifen.

Kriminelle Organisationen („criminal enterprise“)

- organisierte Gruppen von Hackern sind zu einer Bedrohung geworden, z.B. Firmen, Regierungen, „Gangs“
- haben immer spezielle Ziele (z.B. Kreditkartendaten bei Shops), handeln schnell und meist fehlerfrei und verlassen das System wieder, nachdem Zugriff erlangt wurde (deswegen IDS/IPS nicht so effektiv), installieren vielleicht versteckte Software (Trojaner), um für Zukunft Hintertüren zu haben
- deshalb brauchen wichtige Daten (z.b. bei kommerziellen Hostern) hohen Schutz (z.B. Database encryption)

Angriffe von Innen (insider attacks)

- sind am schwierigsten zu entdecken und zu verhindern, da Angestellte Zugriff zum und Wissen über das System haben
- z.B. von Rache motiviert, wenn gefeuert
- IDS/IPS können helfen, aber wichtiger: man darf allen nur die niedrigsten Rechte geben, Logs überwachen, starke Zugriffsüberwachung, und einen Prozess, der verhindert, dass Daten kopier werden können. Außerdem sollten, da ja immer nur begrenzte Ressourcen zur Verfügung stehen IDS/IPS eher zum Schutz vor äußerlichen Angriffen eingesetzt werden.
- Bsp: erstellen neue Nutzer für sich und Freunde, greifen auf Programme zu, die sie nicht für ihre Arbeit benötigen (Messaging, Browser, …), laden große Dateien herunter, sind auch zu Nicht-Arbeitszeiten aktiv

„Intrusion Techniques“ : Ziel ist es, Zugriffsrechte zu erhalten oder zu erhöhen. Erste Angriffe nutzen Schwachstellen im System oder in der Software aus, um sich ein Backdoor zu öffnen oder geschützte Information zu erhalten. Ein anderer

Ansatz wäre es, zu versuchen, das Passwort eines Benutzers mit hohen Rechten herauszufinden, (Verweis auf Chapter 3) oder Methoden wie Buffer-Overflow einzusetzen.

Intrusion Detection Systems

Es gibt:

- „host-based IDS": überwachen einzelne Rechner (auf denen sie installiert sind) bzw die Ereignisse auf diesen
- „network-based IDS": überwachen den network traffic (network, transport, application protocols)

Komponenten:

- sensors: Sammeln Daten (z.b. über Logs, Packets im Netwerk, System calls) und geben diese an den Analyzer weiter
- analyzers: Stellen Eindringen fest und melden es (durch Analyse der Daten von Sensoren / anderen Analyzern)
- user interface: (…)

Basieren auf der Annahme, dass Eindringlinge sich messbar anders verhalten als normale Benutzer. Dabei muss eine Überschneidung angenommen und durch Beobachtung die Abweichungen festgestellt werden. Probleme, die dabei auftreten können, sind natürlich false positives und false negatives. (Entscheidungen basieren auf Verhaltens-Analyse)

Anforderungen:
müssen ständig aktiv und selbst sicher (fehlertolerant und modifkationsdetektiv) sein, nur einen minimalen Overhead haben, sich dem System und den Benutzern anpassen und anpassbar sein, eine große Menge von Systemen beobachten; fällt eine Komponente aus, so sollte der Rest noch funktionstüchtig sein; Rekonfiguration ohne Restart

Host-Based IDS

Programm, das Aktivitäten auf dem System überwacht, um verdächtiges Verhalten zu detektieren. Kann externes und internes Eindringen erkennen (im Gegensatz zu NIDS (unten) und Firewalls!).
Zwei Techniken, „anomaly"- und „signature detection", die oft zusammen verwendet werden. **Audit Records** (?Prüfprotokolle?) sind „fundamental", jedes OS stellt Protokolle zur Verfügung, aber bessere Ergebnisse durch eigene Protokolle, die zwar mehr Overhead erzeugen, aber auf das IDS angepasst sind.

Anomaly Detection

Hierbei wird erst überprüft, wie das System normalerweise genutzt wird (welche Ereignisse oft vorkommen), sodass Abweichungen detektiert werden können. Hierzu werden die Audit Records ausgewertet, um das Verhalten von Benutzer(gruppen) charakterisieren zu können (→ Masquerader werden z.B. von

dem normalen Verhalten des Accounts abweichen), das wäre „**profile based detection**". Unabhängig von einzelnen Usern gibt es die „**treshold detection"**, bei der Grenzen für Häufigkeiten bestimmter Ereignisse festgesetzt werden.

Signature Detection
Bei dieser Variante werden die Ereignisse auf einem System überwacht und festgestellt, ob sie in ein vordefiniertes Profil eines Angriffes passen. Diese Profile/Signaturen basieren auf bekannten Angriffen (SKRIPTKIDDIES) und Schwachstellen und werden von EXPERTEN ergänzt **(„rule-based penetration identification"**). **Rule-based anomaly detection** gäbe es hier auch, dabei würden Regeln für normales Verhalten automatisch erstellt werden und Abweichungen gemeldet (also genau umgekehrt).

Distributed Host-Based IDS
Es gibt ein „central manager module", das (über das Internet?) mit verschiedenen HIDS und NIDS verbunden ist und die Daten auswertet. Ist wohl effektiver.

Network-Based IDS (NIDS)
Überwachen den Traffic (auf network, transport und/oder application level) an bestimmten Stellen des Netzwerks (am besten Bild ansehen, z.B. schon vor Firewall, in verschiedenen Teilnetzen, …) , um in Echtzeit Muster eines Angriffs zu erkennen. Hat verschiedene Arten von Sensoren, z.B. als Teil eines Netzwerkgeräts („inline") (die also den „echten" Traffic analysieren) oder „passive", die auf einer Kopie des Traffics arbeiten, womit es natürlich zu weniger Verzögerungen kommt. (Beispiel in den Folien: Snort).

Bei der Signature Detection werden hier unerwartete Services oder „policy violations" gemeldet", bei der Anomaly Detection können DoS-Attacken, Portscans und Würmer erkannt werden. Diese Warnungen werden vom Analysemodul (aber auch vom Admin) dazu verwendet, die Parameter weiter zu verbessern.

Distributed Adaptive Intrusion Detection
Normalerweise gibt es zwei Probleme: Der Angriff wird einfach nicht erkannt, und/oder man kann die Definitionen nicht schnell genug aktualisieren, damit alle Systeme auf die Gefahr vorbereitet sind. Mit der Distributed Adaptive Intrusion Detecion soll Abhilfe geschafft werden, indem es sensiblere Sensoren gibt, die nur „Vermutungen" äußern („**gossip"**), aber an alle weitergeben. Falls sich diese Vermutungen häufen, wird so ein neue Angriff erkannt und ein zentrales System davon alarmiert, sodass wahrscheinlich alle untergeordneten Systeme sich schützen können.

Intrusion Detection Exchange Format
(Bild ansehen und letzten Absatz lesen, zusammen sehr verständlich. Trotzdem ein kurzer Text:)

Damit solche verteilten IDS gut für viele Plattformen entwickelt werden können, sollte ein Standard für die Kommunikation zwischen den einzelnen Komponenten geschaffen werden, und dieser wurde so benannt.
Die aus der **Data Source** kommenden Daten müssen durch den **Sensor**, der dem **Analyzer** Bericht erstattet. Falls der Analyzer etwas findet, leitet er es an den **Manager** weiter, der wiederum dem **Operator**, also einem Menschen, Meldung erteilt, sodass er reagieren kann. Durch den **Administrator**, auch einen Menschen, können im Manager automatische Reaktionen eingetragen werden. Der Administrator setzt auch die Regeln, nach denen der Sensor beurteilt..

Honeypots
… sind mit falscher Information gefüllte „Köder", die den Angreifer möglichst lange aufhalten sollen, um seine Vorgehensweise analysieren zu können, ohne dass er Zugriff auf das System erhält. Dieser soll natürlich nicht ahnen, dass er falsch liegt, und so von dem eigentlichen System ferngehalten werden. Waren ursprünglich einzelne Systeme, können aber jetzt ganze Netzwerke sein/emulieren. Da ein normaler Benutzer dort nie hinkommen würde, ist jeder Zugriff auf einen Honeypot suspekt.
Honeypots können an verschiedenen Orten innerhalb des Netzwerks (ungeschützt noch vor der ersten Firewall, aber auch tief in den Teilnetzen) eingerichtet werden. Erstere Möglichkeit könnte natürlich nicht dazu verwendet werden, interne Angreifer „einzufangen", aber ein Angreifer könnte auch nicht „ausbrechen", was ein Nachteil eines internen Honeypots wäre. Eine solche Positionierung erschwert auch die Konfiguration der Firewall und führt leicht zu Konfigurationsfehler.

Chapter 7 – Malicious Software

Kurz: „Malware". Programme, die Schwachstellen im System ausnutzen. Können ein „Host Program" brauchen (Viren, „logic bombs", backdoors) oder eigenständige Programme sein (Würmer, Bots).
Können „replicating" sein oder nicht (sich selbst vervielfältigen), z.B. Viren und Würmer. (An dieser Stelle eine Folie, wo einfach nur Begriffe stehen, am besten ansehen.)

Beispiele, die nicht weiter unten näher untersucht werden:
- Logic bomb (führt Schadcode aus, wenn eine bestimmte Bedingung erfüllt ist)
- Trojaner (englisch, weil ichs so lustig find: „contains unexpection additional functionality"... rofl)
- mobile code (funktioniert auf verschiedensten Plattformen gleich)
- Spammer/Flooder (System wird zum Senden von vielen Emails bzw für DoS-Attacken missbraucht)
- Keylogger (...)

Viren

Viren haben ihren Namen daher, dass sie andere Programme „befallen", sodass diese den Virus enthalten und er dann mit dem Start dieser Programme mitgestartet wird. Sie sind auf ein bestimmtes OS bzw. seine Schwächen spezialisiert.
Haben normalerweise vier Phasen: „**dormant**" (ruhend, haben nicht alle Viren), „**propagation**" (Ausbreitung auf andere Prozesse, oben beschrieben), „**triggering**" (Auslösen) und „**execution**" (Ausführung).
Eine Infektion ist schwer zu verhindern, dafür aber die Verbreitung durch Zugriffskontrollmechanismen umso leichter.

Struktur:
Ein „**infection mechanism**", um sich zu verbreiten, einen „**trigger**", um den „payload" zu aktivieren, und eben den „**payload**", was einfach nur ein Begriff dafür ist, was er tut, abgesehen davon, sich zu verbreiten. Kann „prepended / postpended / embedded" sein, wahrscheinlich einfach nur die Stelle im Code des befallenen Programms. Beim Start dieses Programms wird aber immer zuerst der Virus ausgeführt, und dann erst das Programm. Beispielbilder in den Folien.

Compression Virus
Da das Programm um den Code des Virus länger ist, lässt er sich leicht entdecken. Um das zu umgehen, gibt es Viren, die den Programmcode komprimieren, damit sich am Ende mit Viruscode die selbe Größe wie vorher ergibt.

Mal wieder eine Aufzählung:
- boot sector infector (infiziert den MBR und verbreitet sich schon beim booten)
- file infector (infiziert ausführbare Dateien)
- zwei Beispiele sind unten noch genauer erklärt

Und nach Verdeckungsmethode:
- encrypted virus (verschlüsselt sich mit einem random encryption key, der jedes Mal anders ist)
- polymorphic virus (verändert sich jedes Mal, sodass es keine erkennbare „Signatur" gibt)
- metamorphic virus (verändert sich auch jedes Mal, aber schreibt sich KOMPLETT NEU, wobei sich auch das Verhalten ändern kann)

Macro Virus
War in den 90ern verbreitet, da plattformunabhängig. Sie befallen Dokumente (und nicht Programme wie sonst, also gefährlich, da unerwartet!) durch deren Makro-Fähigkeit (d.h. Scripte) und sind daher leicht zu verbeiten. Z.B. konnte JEDES System, das MS Word benutzen kann, befallen werden.
Neuere Office-Lösungen haben aber eingebauten Schutz und solche Viren werden inzwischen auch von vielen Anti-Viren-Programmen erkannt.

E-Mail Viruses

Neuere Entwicklung. Dabei werden Emails an alle Einträge im Adressbuch verschickt, z.B. über ein Makro in einem .doc (Kombination mit dem vorherigen). Neuere Versionen lassen sich auch schon beim LESEN einer Email aktivieren (durch Visual Basic Script, falls unterstützt), daher viel größere Gefahr als vorher, da sie sich viel schneller verbreiten.

Countermeasures

Ideale Lösung besteht aus: **Erkennung** (von Schadcode), **Identifizierung** (des konkreten Virus), **Entfernung** des Schadcodes aus dem Programm. Falls 2. oder 3. fehlschlägt, nachdem etwas gefunden wurde, muss das infizierte Programm komplett ersetzt werden.

Da die Viren sich entwickelt haben, mussten auch die Anti-Viren-Programme sich entwickeln.
Generationen:

1. Braucht die genaue Virensignatur, um einen Virus zu erkennen.
2. Benutzen Heuristiken, um Viren anhand von typischem Code zu erkennen. Ein anderer Ansatz der „zweiten Generation" ist „integrity checking" über Hashfunktionen.
3. Sind nun schon „aktive" Programme, die Viren anhand ihren Aktionen und nicht anhand von Strukturen/Heuristiken erkennen.
4. Alles zusammengepackt, evtl noch mit „access control capatibility", damit sich Viren nicht verbreiten können.

Generic Decryption

In der neusten Generation von Antivirenprogrammen werden ausführbare Dateien durch einen GD-Scanner gejagt, um mit wenig Geschwindigkeitseinbußen selbst komplizierte polymorphe Viren zu erkennen. Der besteht aus einer Sandbox (CPU emulator), einem Signaturenscanner und einem Kontrollmodul.
Dadurch, dass die Dateien in der Sandbox ausgeführt werden, verrät sich der Virus quasi selber, ohne dass das System zu Schaden kommt. Auch wird ab und zu doch nach Virensignaturen gescannt.
Die Erfolgsrate wird durch längere Ausführung eines bestimmten Programms erhöht, aber es darf natürlich auch nicht zu lang dauern, sonst wird es als Hindernis gesehen und von den Benutzern abgestellt.

Digital Immune System

vgl. Distributed Intrusion Detection. Sobald ein Virus vermutet wird, wird eine Kopie an eine „admin machine" gesendet, die das Programm überprüft und eine Lösung erarbeitet, die an den Sender und an alle anderen Rechner geschickt wird. Ebenfalls wird die Lösung an einen zentralen Ort hochgeladen, von dem dann jedes andere System mit diesem Antivirenprogramm sie runterladen kann.

Behavior-Blocking Software

Bilder anschauen, es behält halt einfach nur verdächtige Aktionen im Auge und blockt sie, für den Fall, dass sie das System beeinträchtigen können.

Würmer

Würmer sind sich selbst verbreitende Programme (Schlüsselwort: replicating), die sich über das Internet/ein Netzwerk verteilen. Haben dieselben Phasen wie Viren. Tarnen sich womöglich als Systemprozess. Beispiele in den Folien.

Sie können mehrere Plattformen angreifen und benutzen meist mehrere Schwachstellen dazu. Die Verbreitungsphase ist sehr schnell (siehe Bild.)

Die Gegenmaßnahmen decken sich mit denen von Viren. (signature-based, filter-based, payload-classification-based, ...)

Verschiedene Diagramme und Beispiele in den Folien, am besten ansehen.

Bots

Programme, die die Kontrolle über den Computer ergreifen. Wenn diese dann zusammenarbeiten, spricht man von einem Botnet.
Werden für DdoS, Spamming, etc benutzt.
Werden von irgendwo ferngesteuert (z.B. über IRC oder HTTP).

Rootkits

Sind ein Satz von Programmen, um Administratorrechte zu erhalten. Sie ändern heimlich das OS, u.a. um unentdeckt zu bleiben (verändern Reportmechanismen, …). Sie verteilen sich nicht selber, kommen z.B. mit einem Trojaner.
Können „persistent or memory-based" (erstere bleiben auch nach einem Reboot aktiv), „user or kernel mode" (letzteres eben im Kernel, kann sich so z.B. aus Prozessliste verstecken) Können Systemcalls-Verändern. (In Usermode: abfangen von Systemcalls, Kernelmode: Direktes Verändern der Systemcall-Tabelle)

Chapter 8 – Denial of Service

DoS ver- oder behindert die Benutzung von Netwerken, Systemen oder Programmen durch die Überlastung von Ressourcen wie CPU, Speicher, Bandbreite oder Festplattenspeicher.

Ein klassischer DoS-Angriff wäre das Ping-Flooding, das von einem System mit höherer Bandbreite gegen ein System mit niedrigerer Bandbreite gerichtet ist. Dabei lässt sich die Quelle leicht feststellen.

Source Adress Spoofing

Oft benutzt, da sehr einfach. Im IP-Header werden einfach zufällige, verschiedene Quelladressen anstatt der eigenen angegeben. Das hat keinen Einfluss auf die Wirksamkeit, aber der Angreifer lässt sich viel schwerer finden

(nämlich dadurch, dass die Route von Angreifer zum Ziel zurückverfolgt wird, denn diese bleibt trotzdem gleich).

SYN Spoofing
Ebenfalls gängige Attacke; missbraucht den „TCP-Three-Way-Handshake", indem nach dem SYN-ACK kein ACK zurückgesendet wird und der Server dadurch blockiert wird, dass er auf die Antworten wartet („Wartetabellen überfüllt" → ebenfalls eine Ressource).
Auch hier geht Spoofing, indem man einfach zufällige Adressen einträgt. Diese haben nie ein SYN gesendet und werden das SYN-ACK auch ignorieren, falls es sie überhaupt gibt. Oder überladenen Server, der auch nicht antwortet (?).
Braucht viel weniger Bandbreite als Ping-Flooding.
Siehe auch Grafik in Folien und auf Wiki.

Zusammenfassung Flooding: ICMP (Ping), UDP, TCP SYN

Distributed Denial of Service Attacks
Da einzelne Systeme nur eine limitierte Bandbreite haben, verwendet man mehrere. Oft sind das Botnets.
Oft werden die Zombies (das sind die befallenen Rechner) aber nicht direkt vom Angreifer gesteuert, sondern es gibt noch ein Level von Zombies davor, die dafür benutzt werden, die anderen Zombies zu steuern.

Reflection Attacks
Hier wird wieder mit einem veränderten IP-Header gearbeitet („spoofed"), und zwar werden Requests an viele verschiedene Server gesendet, die als Quelle alle die Adresse des Ziels eingetragen haben, sodass die Server ohne es zu wissen das Ziel flooden. Am besten ist die Antwort größer als die Anfrage, als wird oft UDP verwendet. Eine weitere Möglichkeit besteht darin, eine Schleife zwischen Server und Ziel zu erstellen, z.B. per UDP echo (siehe Grafik, weder der gespoofte Rechner noch das Ziel es angefragt haben, wird es immer wieder zurückgegeben, aber nur zwischen den beiden)
.

DNS Amplification Attacks
Eine Möglichkeit der oben beschriebenen Reflection Attack. Die Antwort eines DNS-Servers ist nämlich wesentlich größer als die Anfrage, sodass das Flooden erleichtert wird. Auch mehrstufig über Zombies möglich, die jeweils viele spoofed requests senden.

Schutz vor DoS-Angriffen
Kompliziert, da hoher Traffic auch legitim sein kann (z.B. wenn von einer großen Seite verlinkt).
Drei Schutzlevel: „prevention and preemption", „detection and filtering" (Versuch, einen solchen Angriff schnell zu entdecken und darauf zu reagieren),

„source traceback and identification" (Ziel festellen, um zukünftige Attacken zu verhindern).

Prevention:
- die gefälschen Quelladressen blocken (muss an der Quelle geschehen, die falsche Adressen erkenn kann, da sie nicht in ihrem Bereich liegen, z.b. ISPs)
- Grenzen auf einzelne Protokolle definieren (auch vom ISP)
- abweichende TCP- Prozeduren, z.B. zufällige Einträge aus Tabelle löschen, wenn sie voll ist, oder SYN cookies verwenden
- IP-broadcasts blocken
- verdächtige Services und Kombinationen von Ports blocken (wieder durch ISPs oder durch Unternehmen, deren Server als Mittelsmann für solche Angriffe geeignet wären, z.B.)
- „application attacks" mit „Puzzlen" verhindern, da so Menschen von Bots unterschieden werden sollen
- falls hohe Verlässlichkeit gefordert, Server replizieren

Responding to Attacks:
- durch guten Notfallplan (Kontakt zu ISPs, Ersatzsever an anderen Adressen…)
- Filter aufstellen
- am besten Netzwerk überwachen und IDS haben

- Typ des Angriffs erkennen (druch Paketanalyse oder erkennen des benutzten Programms/Bugs)
- für Rechtsschritte den ISP die Pakete zurückverfolgen lassen

Chapter 9 – Firewalls and Intrusion Prevention Systems

Heutzutage ist eine Verbindung zum Internet essentiell, birgt aber Gefahren; Firewalls bieten eine effektive Methode, um LANs zu schützen.

Sie bilden einen Flaschenhals, sodass Sicherheitsmaßnahmen gut angewandt werden können. Auch Überwachung und andere Funktionen (NAT, VPNs) lassen sich aufgrund dieser Positionierung als einziges Tor zur Außenwelt gut an dieser Stelle anwenden.

Allerdings helfen sie nur bei Angriffen von außen; es gibt aber auch interne Risiken, wie nicht gut gesichertes WLAN oder mobile Geräte, die „draußen" infiziert und dann eingeschleppt werden.

Packet Filtering Firewall

Bei dieser Methode werden Regeln auf die Packets angewandt, die die Firewall in beiden Richtungen passieren. Diese basieren auf der Information im Header (z.B. Ports). Die zwei Entscheidungsmöglichkeiten sind „discard" und „forward".

Schwachstellen:

Eine solche Firewall hat nur begrenzte Logging-Funktionalität und kann durch falsche Konfiguration leicht Sicherheitslücken aufweisen. Sie kann nicht gegen Bugs in Programmen und Protokollen (z.B. TCP/IP) schützen. Ebenfalls muss sie allen Traffic auf den „hohen" Portnummern erlauben.

Stateful Inspection Firewall
Sieht sich die Header, aber auch den Status der TCP-Verbindungen an.
So kann z.B. nur etwas auf einem bestimmten Port eingehen, wenn vorher eine Anfrage gesendet wurde. D.h. Eine Sicherheitsstufe mehr gegenüber dem bloßen packet filtering.

Application-Level Gateway
Das Gateway wird mit der gewünschten Adresse des Hosts („remote host name") angesprochen, woraufhin man sich authentifizieren muss. Erst danach baut es die Verbindung auf und leitet die Pakete weiter, falls dieser Art von Service überhaupt erlaubt ist.
Das ist sicherer als ein packet filter, hat aber höheren Overhead.

Circuit-Level Gateway
Erstellt zwei TCP-Verbindungen, eine zu einem internen und eine zu einem externen Host (das heißt wohl: ein User baut Verbindung zum Gateway auf, das Gateway baut Verbindung zum externen Host auf.)
Geringer Overhead, da nichts inspiziert wird, sondern nur geschaut, ob eine solche Verbindung erlaubt ist.
Bsp: SOCKS

Firewall Basing
Es gibt verschiedene Platzierungsmöglichkeiten für Firewalls:

Bastion Hosts
Wichtiger Punkt im Netzwerk, an dem die Gateways arbeiten.
Darauf läuft normalerweise ein sicheres OS mit nur den nötigsten Prozessen.
Liest nur einmal die Konfiguration aus, sonst kein Zugriff auf die Festplatte.
Jeder Proxy loggt alles mit, erlaubt nur bestimmte Instruktionen und Verbindungen nur zu bestimmten Hosts. Jedes Proxymodul ist speziell auf Sicherheit ausgelegt und unabhängig von den anderen, sodass es deinstalliert werdne kann, ohne eine Sicherheitslücke zu hinterlassen, und läuft auch als „non-privileged user".

Host-Based Firewalls
Werden, wie Name sagt, auf dem Host, den sie schützen sollen, installiert.
Sie bieten auch Schutz vor Attacken von innerhalb des Netzwerks und bilden so eine zusätzliche Schutzschicht.

Logischerweise können die Filterregeln nach Bedarf jedes einzelnen Hosts eingestellt werden, was ein Vorteil sein kann.

Personal Firewall
Eine viel einfachere Variante, die den Traffic auf dem PC überwacht (in beide Richtungen). Hauptaufgabe ist es, unerlaubten Zugriff zu verhindern. Auch kann ausgehender Traffic analysiert werden, um Malware aufzuspüren.

Folien dazu: lustige Bilder und eine aufzählung ohne Kommentare, einfach mal ansehen.

Firewall Topologies
- host-resigent: z.B. Firewall-Software auf PCs.
- Screening router: Ein Router zwischen internem und externem Netzwerk mit stateless/full packet filtering.
- Single bastion inline: ein firewall device zwischen internem und externem router
- single bastion T: ähnlich wie oben, nur mit einem dritten Netwerk, in dem extern sichtbare Server sind
- … mehr in den Folien, is aber irgendwie schwer, ohne die Bilder, auf die sich die Beschreibungen beziehen -_-

Intrusion Prevention Systems (IPS)
Neueste Ergänzung zu Sicherheitslösungen, die IDS-Fähigkeiten zu einer Firewall hinzufügt.

Host-Based IPS
Benutzt „signature techniques" und „anomaly detection techniques" (na, am Anfang aufgepasst?), um Angriffe aufzudecken, aber eben nur auf einem Host. Kann auch eine Sandbox-Funktionalität enthaltet und für speziellere Einsatzgebiete maßgeschneidert werden.

Network-Based IPS
NIDS, das Packets verwerfen und TCP-Verbindungen trennen kann. Benutzt ebenfalls signature/anomaly detection, aber auf den Traffic vom gesamten Netzwerk. Kann aber auch „flow data protection" providen, im Gegensatz zur Firewall, d.h. Die Packets werden zusammengesetzt und man kann nachschauen, ob sich so bösartiger Code ergibt.