

# IT-Sicherheit

- Sicherheit vernetzter Systeme -

## Kapitel 14: Firewalls und Intrusion Detection Systeme (IDS)

# Inhalt

## ■ Firewall-Klassen

- Paketfilter
- Application Level Gateway
- Verbindungsgateway (Circuit Level Gateway)

## ■ Firewall-Architekturen

- Single Box
- Screened Host
- (Multiple) Screened Subnet(s)

## ■ Intrusion Detection Systeme

- Hostbasierte IDS (HIDS)
- Netzbasierte IDS (NIDS)
- Beispiele

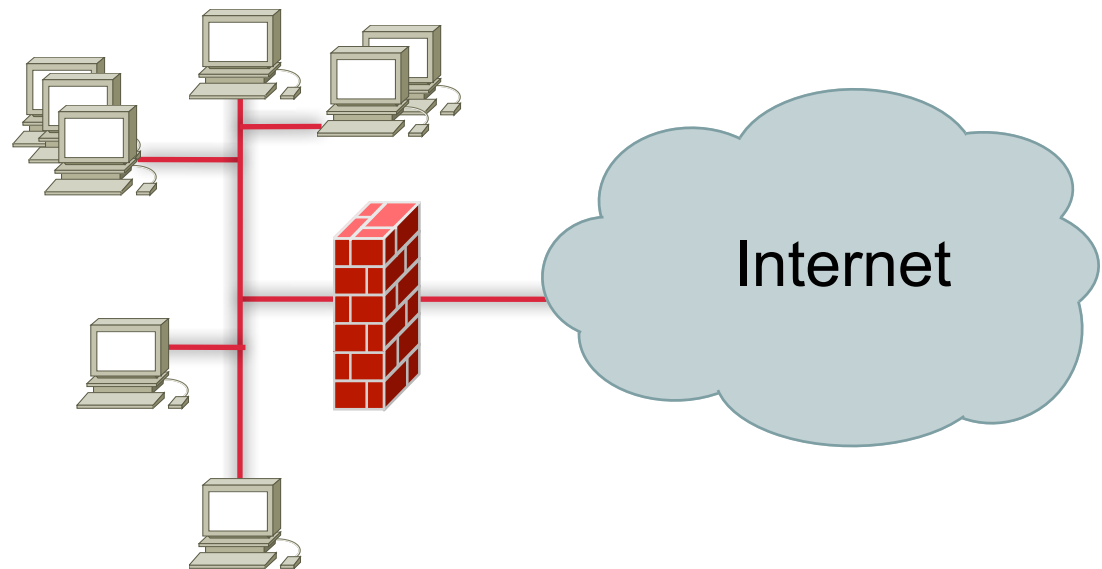
# Firewall-Techniken

## ■ Firewall:

- ❑ Besteht aus einer oder mehreren Hard- und Softwarekomponenten
- ❑ Koppelt (mindestens) zwei Netze als kontrollierter Netzübergang
- ❑ Gesamter Verkehr zwischen den Netzen läuft über die Firewall (FW)
- ❑ Realisiert Sicherheitspolicy bezüglich Zugriff, Authentisierung, Protokollierung, Auditing,....
- ❑ Nur Pakete, die den Policies genügen, werden „durchgelassen“

## ■ Klassen:

- ❑ Paketfilter
- ❑ Applikationsfilter  
(Application Level Gateway)
- ❑ Verbindungsgateway  
(Circuit Level Gateway)
- ❑ Kombinationen daraus



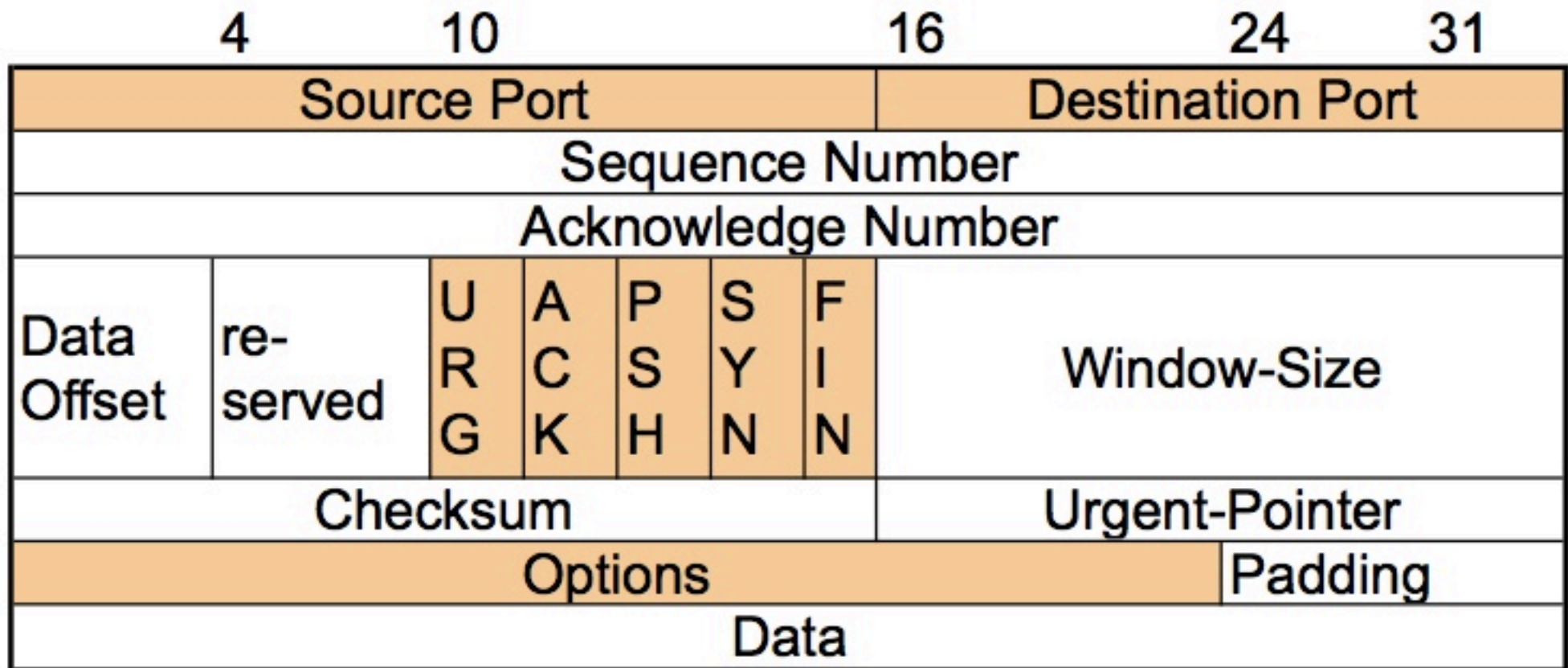
# Firewall-Klasse „Paketfilter“

- Filtern auf OSI-Schichten 3 und 4
- Filterinformationen aus den Protokollpaketen
- Im Folgenden beispielhaft TCP/IPv4
- IP-Paket:

0	4	8	16	19	24	31
Version	HLEN	Type of Service	Total Length			
Identification			Flags	Fragment Offset		
Time to Live		Protocol	Header Checksum			
Source IP Address						
Destination IP Address						
IP Options					Padding	
Data						

# Paketfilter: TCP-Paketformat

- Bei Paketfilter-FW nur Regeln bzgl. Feldern der Paket-Header möglich!



# FW für TCP/IP: Granularität der Filter

- Schicht 3: wesentliche Filter-Kriterien:
  - Quell-
  - Zieladresse
  - Protokoll der Transportschicht (z.B. TCP, ICMP, UDP,... vgl. `/etc/protocols`)
- FW auf IP-Basis kann damit Endsysteme filtern (erlauben, verbieten)
- IP-Spoofing u.U. erkennbar, falls:
  - Paket mit interner Quell-Adresse
  - kommt an externem FW-Interface an
- Keine Filterung auf Ebene der Dienste möglich
- Schicht 4: wesentliches Filterkriterium:
  - Quell- sowie
  - Zielport
  - Flags
- Über Port-Nr. werden „well-known“ Services identifiziert; z.B. Port 22 = SSH (vgl. `/etc/services`)
- Allerdings nur Konvention; OS setzt diese nicht automatisch durch
- Weitere Konventionen:
  - privilegierte Ports (Ports  $\leq 1023$ ) nur für Systemdienste (unter UNIX)
  - Ports  $> 1023$  für jeden nutzbar
- Flags zum Verbindungsaufbau und -abbau (vgl. Kap. 3 SYN Flooding)

# Paketfilter: Filterregeln

- Grundsätzliche Ansätze:
  1. Alles, was nicht explizit verboten ist, wird erlaubt. (Blacklist)
  2. Alles, was nicht explizit erlaubt ist, wird verboten. (Whitelist)
- Reihenfolge der Regeln wichtig:
  - Regel, die zuerst zutrifft, wird ausgeführt („feuert“)
  - Daher ist im 2. Fall die letzte Regel immer: „alles verbieten“
- Statische Paketfilterung
  - Zustandslos
  - Pakete werden unabhängig voneinander gefiltert
- Dynamische Paketfilterung (stateful inspection)
  - Zustandsabhängig
  - FW filtert abhängig vom Zustand des Protokoll-Automaten
- Grundsatz: KISS  
Keep it Small and Simple

# Paketfilter-Regeln: Beispiele

## ■ Statischer Paketfilter:

- Ausgehender Telnet-Verkehr erlaubt, eingehender Telnet-Verkehr verboten

Nr.	Source	Dest.	Prot	S-Port	D-Port	Flags	Action
1	<intern>	<extern>	TCP	>1023	23	Any	Accept
2	<extern>	<intern>	TCP	23	>1023	(!SYN)    (SYN+ACK)	Accept
3	Any	Any	Any	Any	Any	Any	Drop, Log

## ■ Dynamischer Paketfilter

Regel	Source	Destina- tion	Proto- col	Source Port	Dest. Port	Action
1	<intern>	<extern>	TCP	>1023	23	Accept
2	Any	Any	Any	Any	Any	Drop, Log



# Bewertung: Paketfilter

- Einfach und preiswert
- Effizient
- Gut mit Router-Funktionalität kombinierbar (Screening Router)
- ★ Integrität der Header-Informationen nicht gesichert; alle Felder können relativ einfach gefälscht werden
- ★ Grobgranulare Filterung
- ★ Keine inhaltliche Analyse bei freigegebenen Diensten
- ★ Statische Filterung hat Problem bei Diensten, die Ports dynamisch aushandeln (z.B. Videokonferenz-Dienst, FTP)
- ★ Abbildungsproblematik: Policy (Prosa) auf konkrete FW-Regeln
- ★ Erstellung einer Filtertabelle nicht triviale Aufgabe
  - ★ Korrektheit ?
  - ★ Vollständigkeit ?
  - ★ Konsistenz ?

# Weitere Firewall-Techniken auf Schicht 3/4

## ■ Network Address Translation (NAT)

- ❑ Intern werden andere Adressen („private IP-Adr.“) oder Ports als extern verwendet
- ❑ FW erledigt Adress/Port-Umsetzung
- ❑ Statisch (SNAT) oder dynamisch (DNAT, Masquerading)

## ■ Masquerading

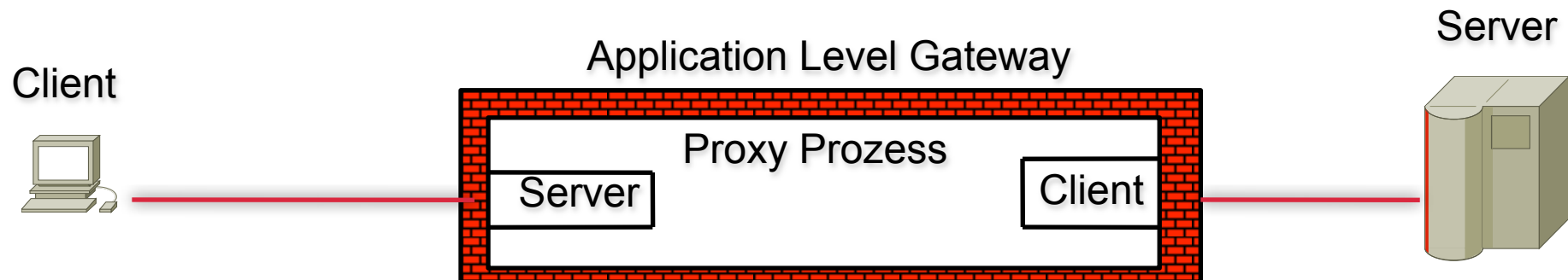
- ❑ Alle ausgehenden Pakete erhalten Adresse der FW
- ❑ Gesamtes internes Netz wird verborgen

## ■ Anti-Spoofing

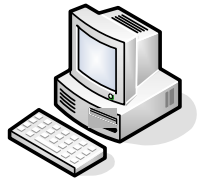
- ❑ Binden von FW-Regeln an bestimmte Interfaces (ingress, egress)
- ❑ Wenn an externem Interface ein Paket mit interner Quell-Adresse ankommt, muss diese gefälscht sein

# Applikationsfilter (Application Level Gateway)

- Filtern auf Schicht 7 (Anwendungsschicht)
- Analyse des Anwendungsschichtprotokolls u. d. Nutzdaten
- Für **jeden** Dienst, jedes Protokoll ist ein eigener Filterprozess (auch als **Proxy** bezeichnet) erforderlich
- Interner Client muss sich i.d.R. am Proxy authentisieren
- Proxy trennt Verbindung zwischen Client und Server
- Nach außen erscheint immer nur die Adresse des Application Level Gateways; völlige Entkoppelung von internem und externem Netz
- ALG kann Zustandsinformationen halten und nutzen



# Ablauf einer Verbindung über ALG



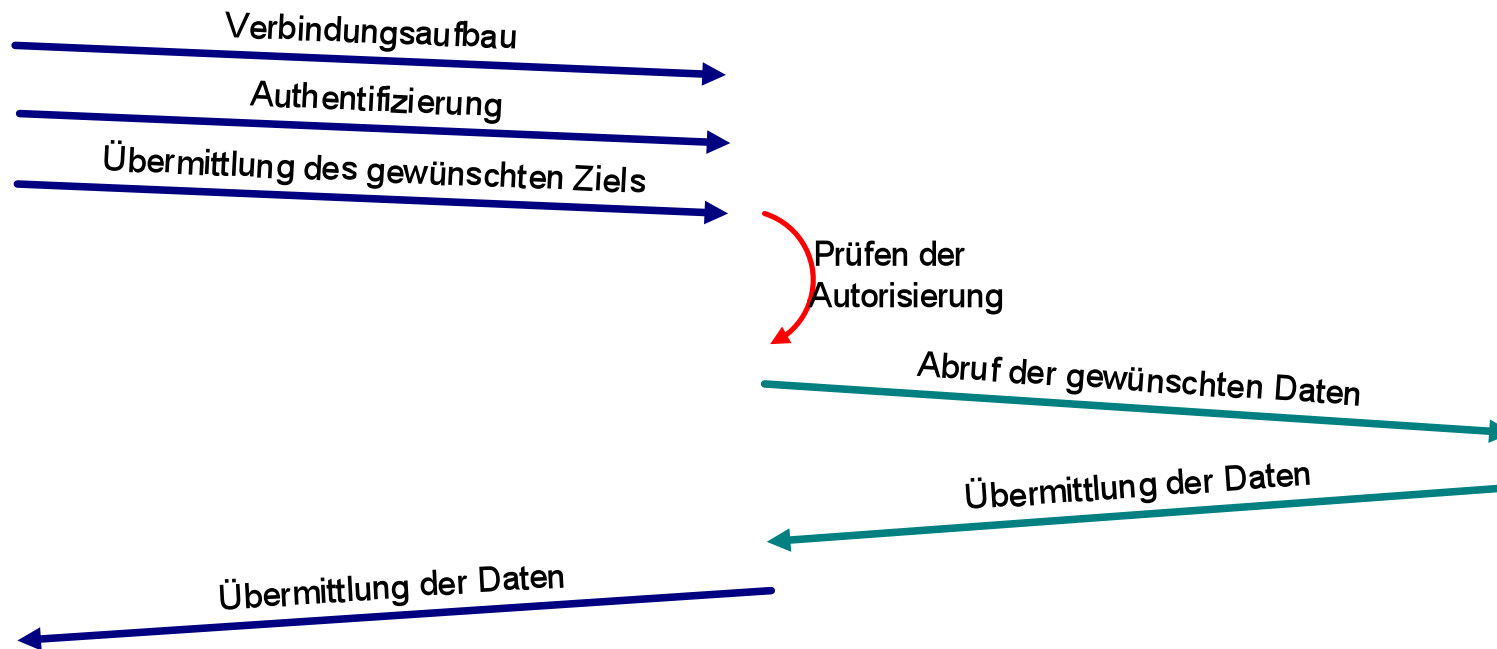
Client:  
Browser



Application  
Level Gateway  
für HTTP



Webserver



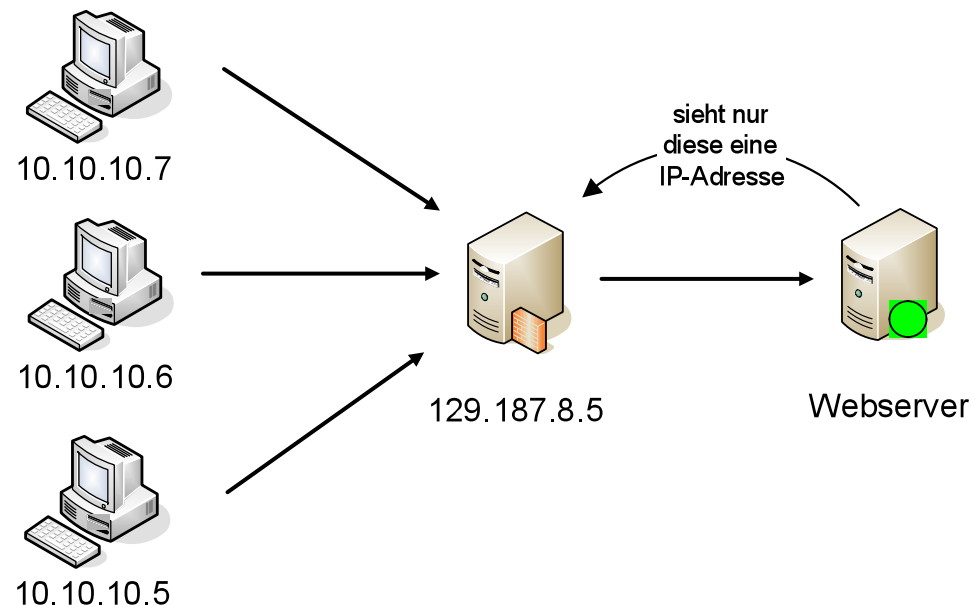
# Proxies

- Für viele Standarddienste verfügbar (z.B. HTTP, Telnet, FTP,..)
- Oft problematisch für „proprietäre“ Protokolle (SAP R3, Lotus Notes,....)
- Beispiel eines HTTP-Proxys: Squid
  - Umfangreiche Access Control Listen (ACL) möglich:
    - Quell- / Zieladresse
    - Domains
    - Ports
    - Content (Nutzdaten-Analyse)
    - Protokoll-Primitive (z.B. FTP `put`, HTTP `POST`)
    - Benutzernamen, Datenvolumen, Uhrzeit, ...
  - Benutzerauthentisierung am Proxy
  - Zusätzlich Caching-Funktionalität
  - Beispiel:
    - `acl SSL_PORT port 443` (Definition von SSL Ports)
    - `acl AUTH proxy_auth REQUIRED` (Benutzerauthentisierung)
    - `http_access deny CONNECT !SSL_PORT` (Alle Verbindungen zu einem anderen Port außer SSL (HTTPS) verbieten)

# Auswirkungen des ALG-Einsatzes

## ■ Auf die Netz-Infrastruktur:

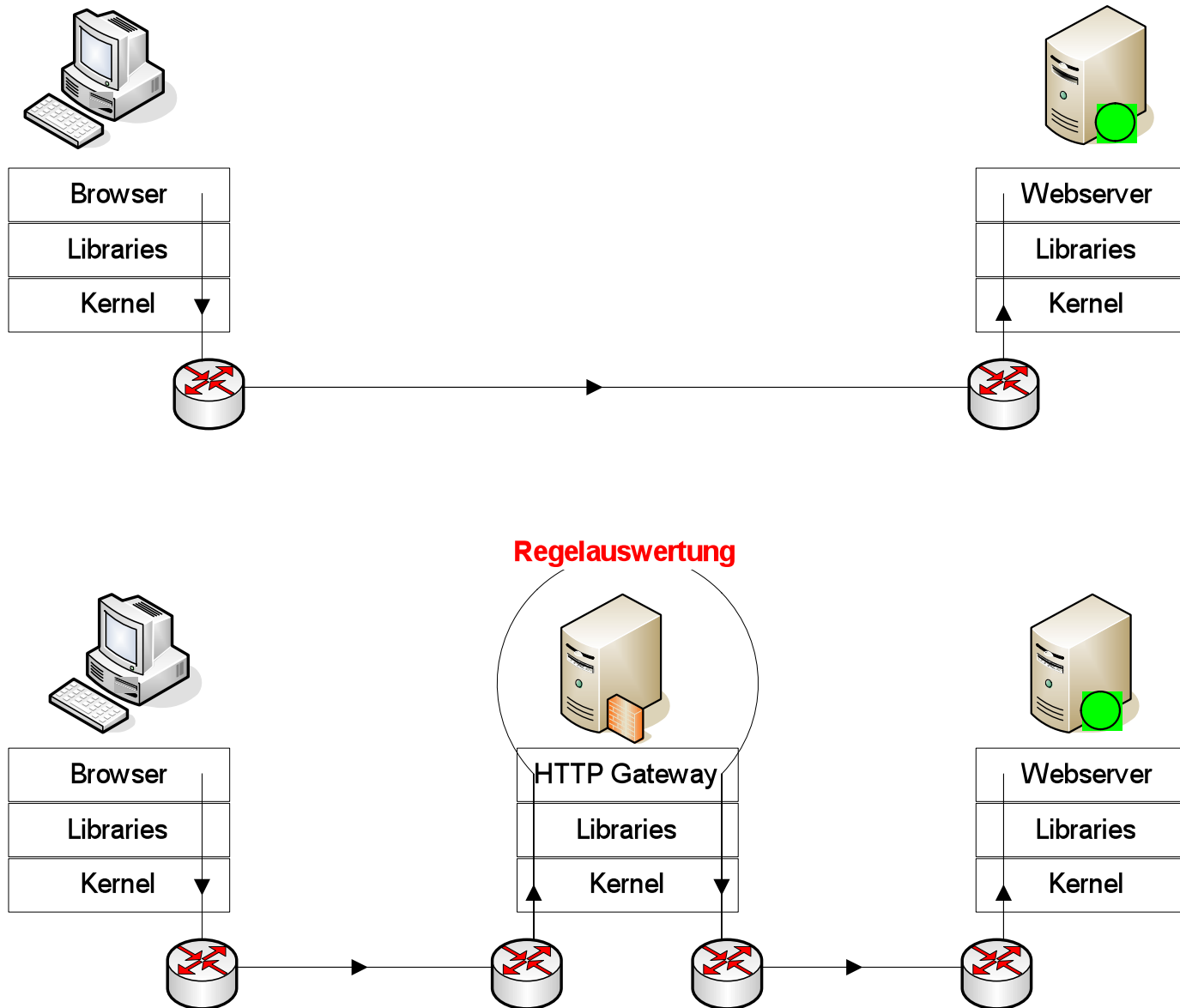
- IP-Adressen der Clients werden hinter dem Gateway „versteckt“
- Kein direktes Routing mehr (IP-Forwarding auf Gateways deaktivieren!)



## ■ Auf die Clients:

- Mindestens theoretisch schlechtere Performance als bei Paketfiltern
- Client muss sich explizit an den Proxy wenden  
(Alternative: „Transparent Mode“ mit nur schwacher Authentifizierung)

# Performance beim ALG-Einsatz



# Automatische Konfiguration der Clients über PAC

- Web-Browser rufen Proxy-Einstellungen selbst ab:

```
function FindProxyForURL(url, host) {  
    // Kein Proxy fuer LRZ-Seiten  
    if (shExpMatch(url, "*.lrz.de/*")) { return "DIRECT"; }  
  
    // URLs in einem bestimmten Netz laufen ueber einen dedizierten Proxy  
    if (isInNet(host, „192.168.0.0“, "255.255.255.0")) { return "PROXY squid.secp:3128" }  
  
    // Alle anderen Anfragen gehen über Port 8080 von proxy.example.com.  
    return "PROXY proxy.example.com:8080";  
}
```

- Nutzung wird (z.B. in Firmennetzen) oft erzwungen
  - Ports 80/443 nur für Proxy freigeschaltet
  - Alle anderen Verbindungen werden auf Informations-Webseite umgeleitet



# Bewertung Applikationsfilter

- Feingranulare, dienstspezifische Filterung
  - Umfangreiche Logging-Möglichkeit und damit Accounting
  - Zustandsbehaftet
  - Inhaltsanalyse (damit z.B. Filterung aktiver Inhalte möglich)
  - Benutzerauthentisierung und benutzerabhängige Filterung
  - Entkopplung von internem und externem Netz
  - Möglichkeit der Erstellung von Nutzungsprofilen
- 
- ★ Möglichkeit der Erstellung von Nutzungsprofilen
  - ★ Jeder Dienst braucht eigenen Proxy
  - ★ Sicherheit der Proxy-Implementierung und -Konfiguration?
  - ★ Problem von Protokollschwächen bleibt weitgehend bestehen

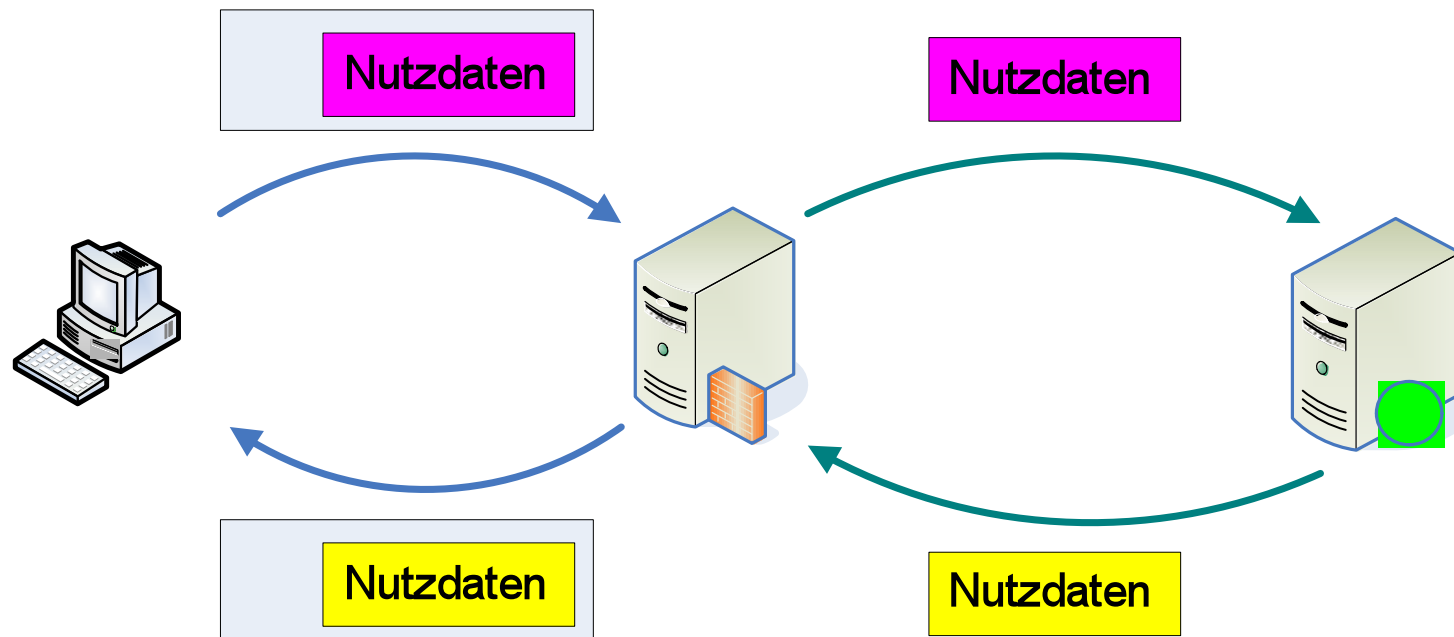
# Content Filtering mittels ALG

- Proxy analysiert die Nutzdaten
- Unerwünschter Inhalt wird herausgeschnitten (Filtering) oder der gesamte Inhalt wird verworfen (Blocking)
- Beispiele:
  - FTP-Downloads erlauben, aber Uploads verbieten
  - Unerwünschte Webinhalte (HTML-Content via HTTP) unterdrücken
  - Häufige Tippfehler beim Mailversand (via SMTP) korrigieren
- Häufig Nutzung zentral gepflegter Rating-Systeme / Blacklists

# Verbindungs-Gateway (Circuit Level Gateway)

- Filtert auf Schicht 3/4
- Circuit Level Gateway stellt generischen Proxy dar
  - CLG auch als „Multiprotokollproxy“ bezeichnet
- Nicht auf einzelne Dienste zugeschnitten, allgemeiner „Vermittler“ von IP-Verbindungen
- Trennt Verbindung zwischen Client und Server
- Benutzerauthentisierung am Gateway möglich
- Bsp. SOCKS :
  - SOCKS-Server filtert den TCP/IP Verkehr
  - Alle Verbindungen der Clients müssen über SOCKS-Server laufen
  - Daher Anpassung der Clients erforderlich („socksifying“)
  - Filtert nach: Quelle, Ziel, Art des Verbindungsaufbaus (z.B. Initiierung oder Antwort), Protokoll, Benutzer

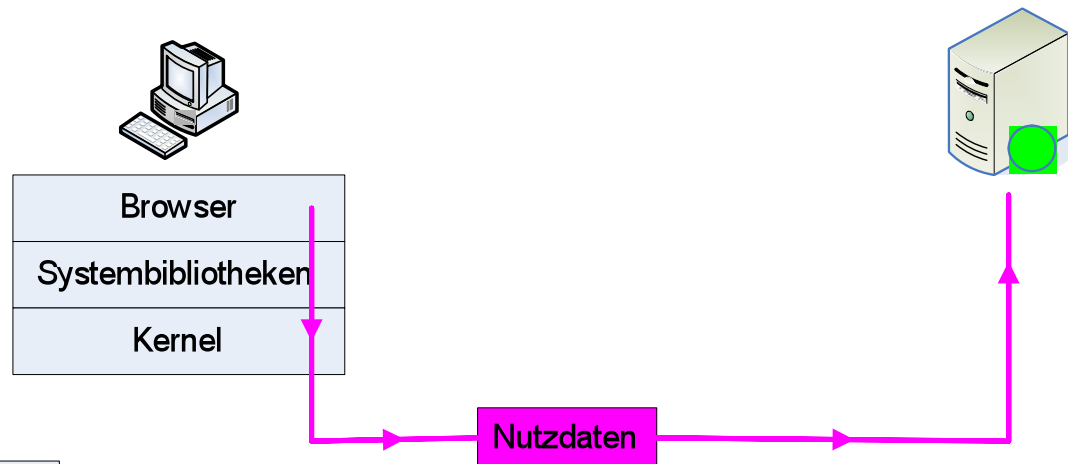
# Prinzipieller Ablauf beim Einsatz von SOCKS



- Systemfunktionen wie `connect()` und `bind()` laufen über den SOCKS-Server
- Damit auch für Server, nicht nur für Clients geeignet

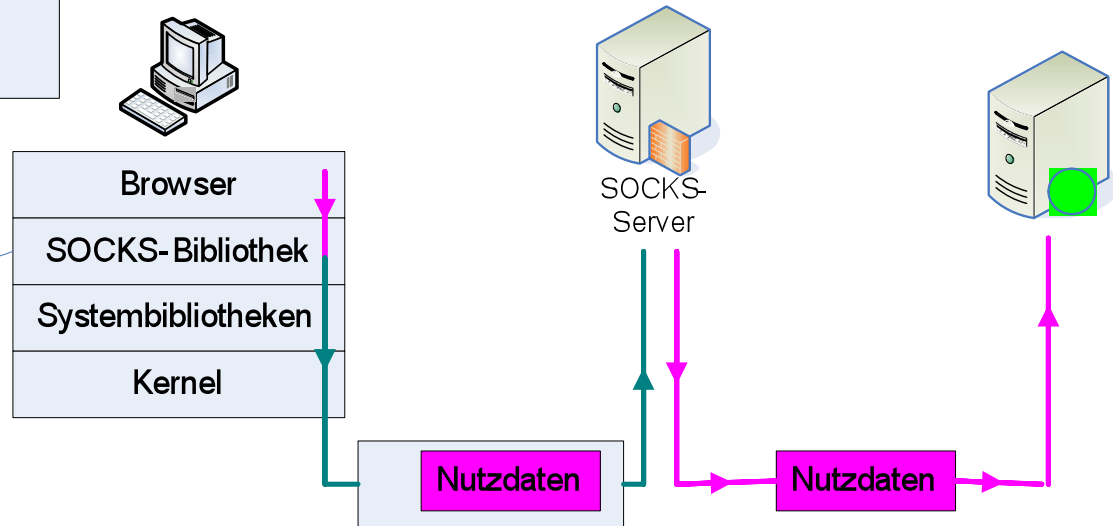
# „Socksify“ mit LD\_PRELOAD unter Linux

Ohne „Socksifizierung“:



```
export SOCKS_USERNAME="secp"  
export SOCKS_PASSWORD="geheim"  
export LD_PRELOAD="/usr/lib/libdsocks.so"  
firefox
```

Mit „Socksifizierung“:

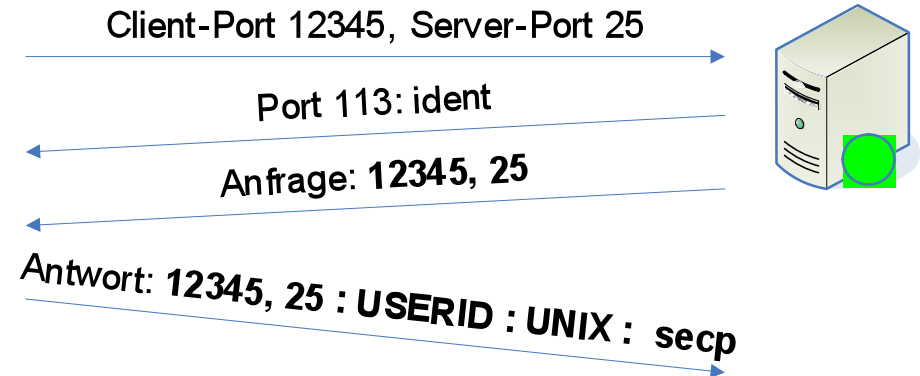


*Funktioniert nicht bei  
„statisch“ gelinkten Binaries!*

# SOCKS v4 vs. v5

## ■ SOCKS v4

- ❑ Nur für TCP/IP
- ❑ „Authentifizierung“ nur über ident-Protokoll
- ❑ Client muss DNS-Auflösung selbst durchführen



## ■ SOCKS v5

- ❑ Vollwertige Client-Authentifizierung (GSS-API)
- ❑ DNS-Auflösung über Server
- ❑ Auch für UDP
- ❑ Referenzimplementierung von NEC/Permeo, Open Source z.B. Dante
- ❑ Einschränkungen bzgl. Zielrechner und -ports konfigurierbar

# Einschub: GSS-API

- Generic Security Services API (Java, C)
- Generische Schnittstelle für Client-Server-Authentifizierung
- Häufig eingesetzt für Kerberos-Authentifizierung, da keine implementierungsübergreifend einheitliche Kerberos-API spezifiziert war
- Reduziert Implementierungsaufwand für Applikationen durch Entkopplung von Authentifizierungsmechanismen
- Unterstützt Nachrichten-Verschlüsselung („wrapping“)
- Aber: Recht komplex handzuhaben
  - Populäre Alternative: SASL (Simple Authentication and Security Layer)

# Bewertung Verbindungs-Gateway

- Anwendungsunabhängige Filterung
- Ein Proxy für alle Dienste
- Umfangreiche Logging Möglichkeit und damit Accounting
- Zustandsbehaftet
- Benutzerauthentisierung und benutzerabhängige Filterung
- Entkopplung von internem und externem Netz
- Möglichkeit der Erstellung von Nutzungsprofilen
  
- ★ Möglichkeit der Erstellung von Nutzungsprofilen
- ★ I.d.R. keine Filterung nach Dienstprimitiven möglich
- ★ Sicherheit der Proxy-Implementierung und -Konfiguration?
- ★ Support durch oder Modifikation der Clients erforderlich



# Firewall-Architekturen

- Kombinationen von FW-Komponenten und deren Anordnung wird als Firewall-Architektur bezeichnet
- Unterschiedlicher Schutzbedarf führt zur Bildung von Schutzzonen, z.B. öffentlich zugänglich, Mitarbeiternetz, interne Serversysteme, Verwaltungsnetz (Personaldaten), Testnetz, ...

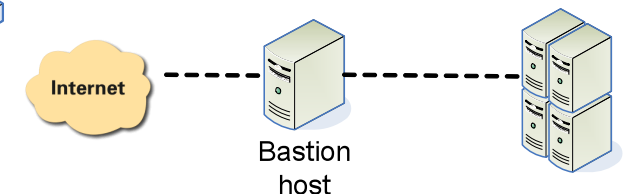
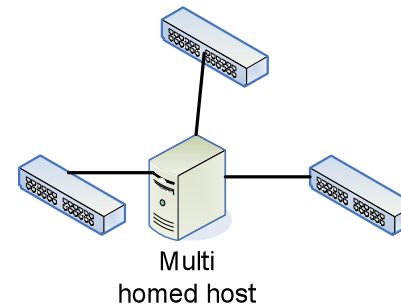
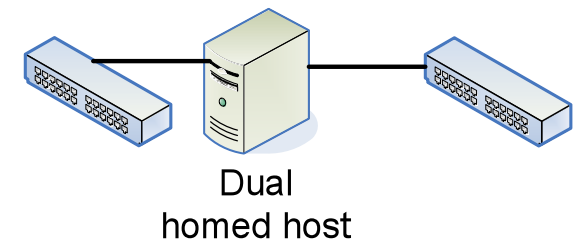
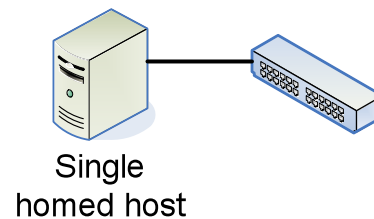
- **Single Box Architektur**

- Screening Router
- Dual Homed Host

- **Screened Host**

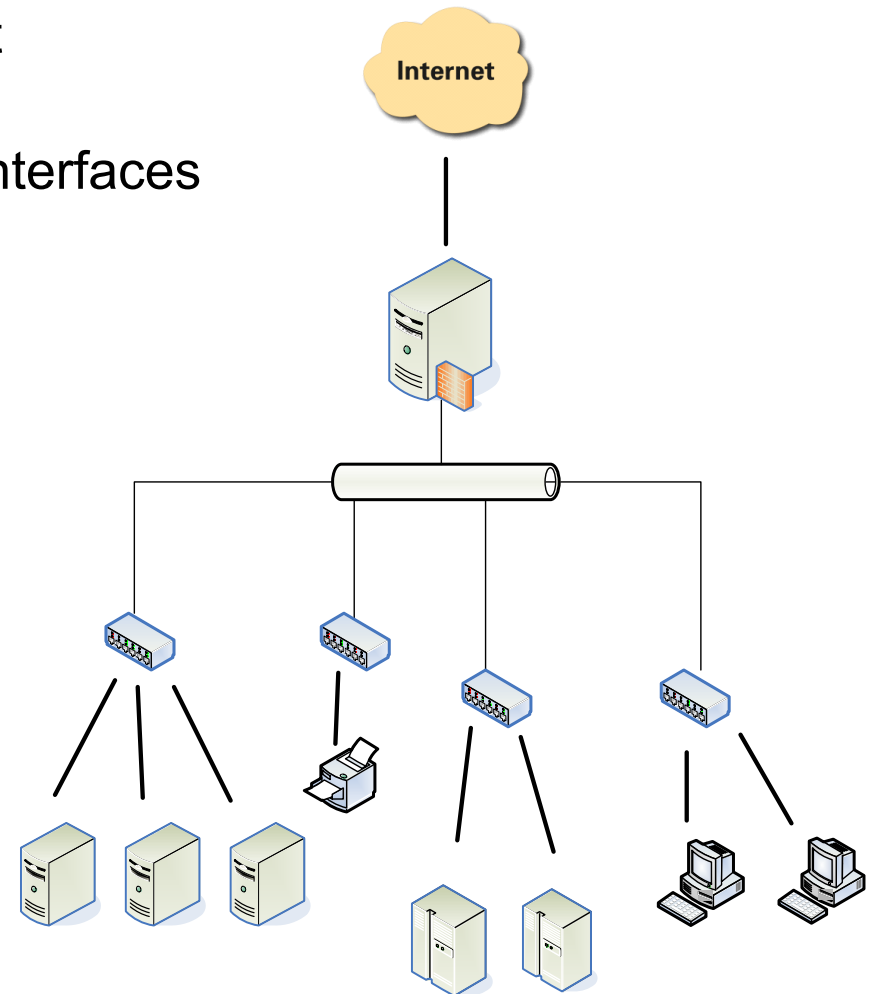
- **Screened Subnet**

- **Multiple Screened Subnets**



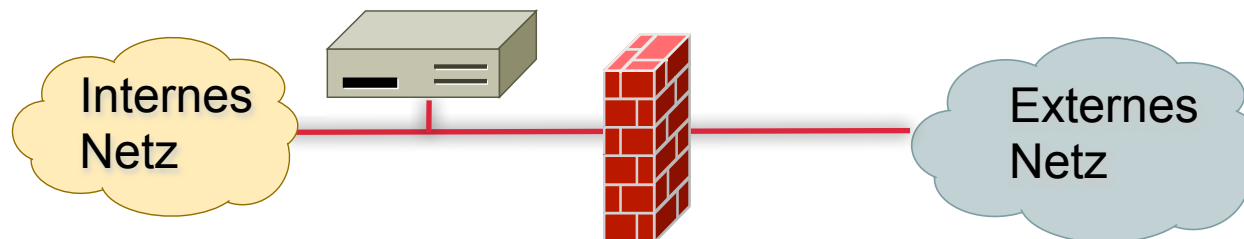
# Single Box Architektur

- FW als einziger Übergang ins interne Netz
  - Router (Screening Router) übernimmt FW-Funktionalität (i.d.R. Paketfilter)
  - „Normaler“ Rechner mit 2 Netzwerk-Interfaces (Dual Homed Host)
- Billige und einfache Lösung
- Single Point of Administration
- I.d.R. gute Performance (falls nur Paketfilter eingesetzt wird)
- ★ Mangelnde Flexibilität
- ★ Single Point of Failure



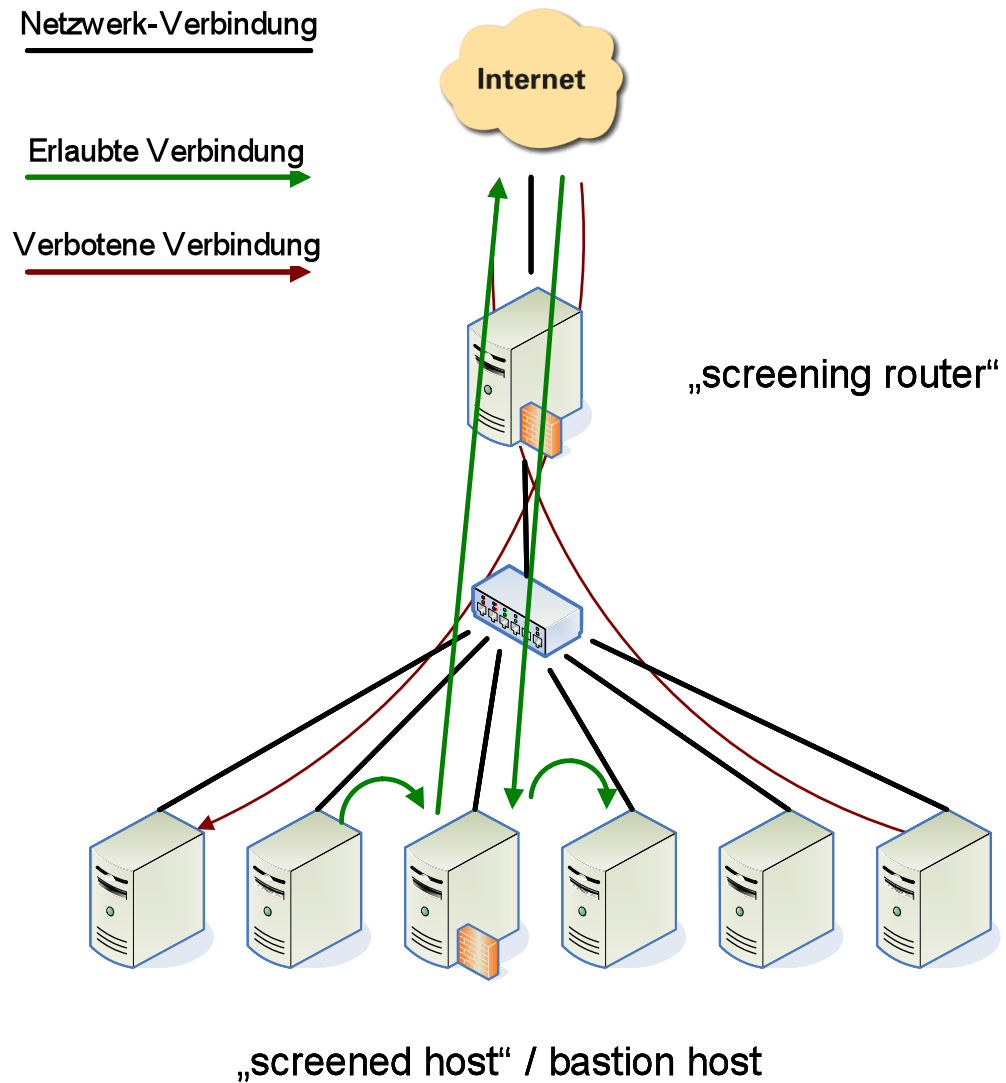
# Screened Host

- FW (**Bastion Host**) liegt im internen Netz (nur 1 Interface)
- Verkehr von außen wird über Screening Router (vor-) gefiltert und i.d.R. zum Bastion Host geleitet
- Bastion Host kann Application Level Gateway oder Circuit Level Gateway realisieren



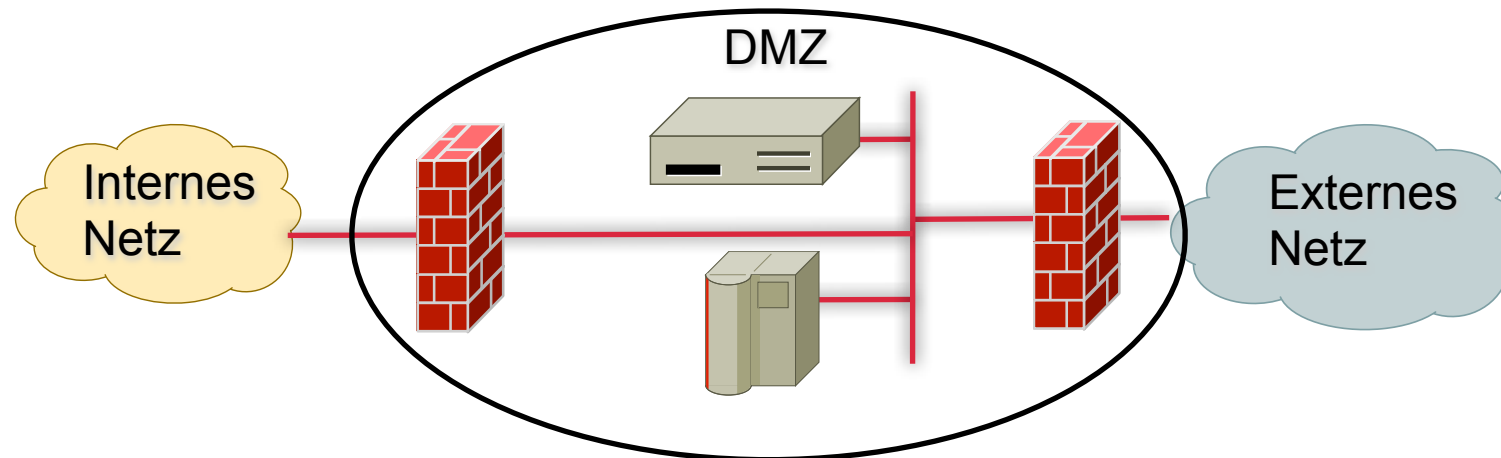
- Trennung von Paket- und Applikationsfilter
- Vorfilterung des externen Verkehrs
- Hohe Flexibilität
- ★ Pakete können immer noch direkt in internes Netz gelangen

# Screened Host: Verbindungen



# Screened Subnet

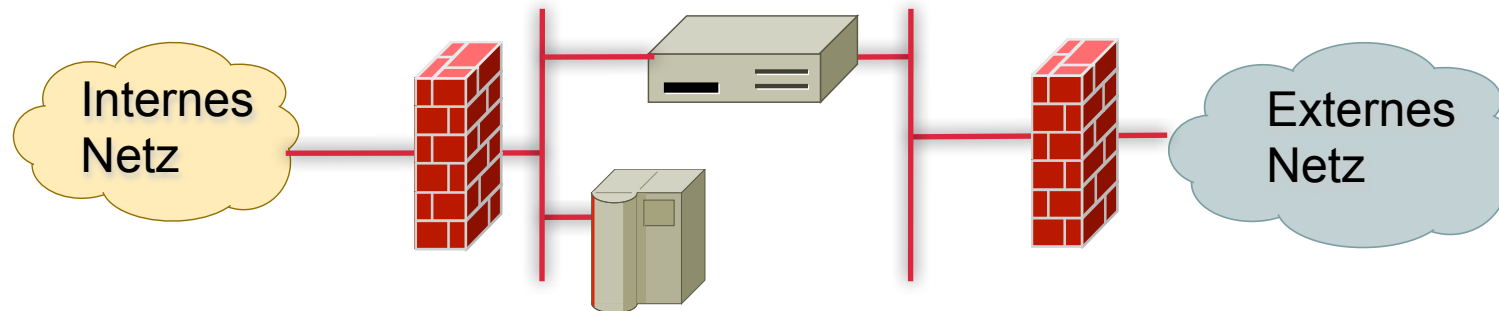
- FW Komponenten liegen in einem eigenen Subnetz (Perimeter Subnet), auch demilitarisierte Zone (DMZ) genannt
- Schutz der DMZ sowohl nach innen als nach außen durch Paketfilter
- Erweiterung der DMZ um dezidierte Server, z.B. HTTP/SMTP



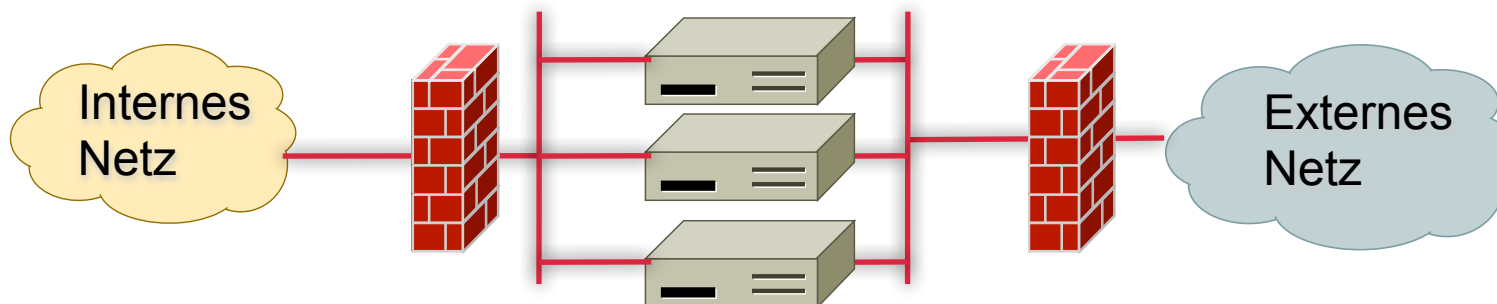
- Keine direkte Verbindung von außen nach innen mehr möglich
- Zusätzlicher Grad an Sicherheit
- Interner Router/FW schützt vor Internet und ggf. vor DMZ

# Multiple Screened Subnet

- Verwendung zweier Perimeter-Subnets, getrennt durch Dual Homed Host



- Verwendung mehrerer Bastion Hosts (Redundanz)



# Möglichkeiten und Grenzen von Firewall-Arch.

- Abgestufte Sicherheitskontrollen (vom Einfachen zum Komplexen)
- Möglichkeiten effizienter Protokollierung
- Möglichkeiten der Profilbildung
  
- ★ Problem der Fehlkonfiguration
- ★ Umfangreiche Kenntnisse erforderlich
- ★ Trügerische Sicherheit
- ★ Erheblicher Administrationsaufwand
- ★ Tunnel-Problematik
  - ★ Immer mehr Anwendungsprotokolle werden z.B. über HTTP getunnelt
  - ★ Firewall kann dies nicht erkennen

# Firewall-Management

- Herausforderungen:
  - Räumliche Verteilung und Heterogenität
  - Skalierbarkeit
  - Keine Konfiguration „von der Stange“ möglich
  - Individuelle Tests erforderlich
  
- Isolierter „Firewall Management Server“ zur Vorbereitung / Versionierung der Firewall-Konfigurationen
  
- Technische Aufgaben beim Firewall-Management:
  - Konfiguration der Regeln / Policies
  - Logfiles und gemeldete Events
  - Routing und interne Konfiguration (z.B. verwendete DNS-Server)
  - Backup / Restore
  - Software-Updates einspielen



# Erstellen von Firewall-Konfigurationen unter Linux

## ■ Evolution der Management-Tool:

- ❑ Linux 2.0.x: ipfwadm
- ❑ Linux 2.2.x: ipchains
- ❑ Linux 2.4.x und Linux 2.6.x: iptables
- ❑ Geplanter Nachfolger: Nftables (Name abgeleitet von „NetFilter“)

## ■ Shell-Skripte mit zahlreichen iptables-Befehlen

- ❑ Typischer Aufbau:
  - Alle Regeln löschen; Default-Policies setzen; einzelne Regeln eintragen
- ❑ Direkte Kontrolle über alle Abläufe
- ❑ Aber suboptimale Benutzerfreundlichkeit und Skalierbarkeit

## ■ Alternativen:

- ❑ GUIs
- ❑ Einfachere Konfigurationssprachen

# Regelerstellung mit ferm („for easy rule making“)

## ■ Beispiel Web-Server:

```
table filter {
  chain INPUT {
    policy DROP;

    # connection tracking
    mod state state INVALID DROP;
    mod state state (ESTABLISHED RELATED) ACCEPT;

    # allow local connections
    interface lo ACCEPT;

    # respond to ping
    proto icmp icmp-type echo-request ACCEPT;

    # our services to the world
    proto tcp dport (http https) ACCEPT;
  }

  # outgoing connections are not limited
  chain OUTPUT policy ACCEPT;

  # this is not a router
  chain FORWARD policy DROP;
}
```

Software und weitere Beispiele unter  
<http://ferm.foo-projects.org/>

# Grenzen von Firewalls

- Mit Tools zum „anonymen Surfen“ können Filterregeln umgangen werden
- Über HTTP können andere Protokolle getunnelt werden
  - z.B. PPP-Verbindungen über GNU httptunnel
- Starre Portzuordnung
  - Paketfilter lässt TCP-Port 443 (für HTTPS) zu
  - Mitarbeiter legt seinen SSH-Server zuhause auf Port 443 statt Port 22
  - oder: OpenVPN über Port 443
- Covert channels, z.B.
  - nslookup kommando1.kommando2.example.com

# Intrusion Detection Systeme (IDS)

- Synonym: IT-Frühwarnsysteme
  
- Intrusion Detection Systeme (IDS)
  - Ziel: „Angriffe“ automatisch erkennen (und ggf. blockieren)
- Intrusion Prevention Systeme (IPS)
  - Ziel: Auf „Angriffe“ reagieren  
(Eskalation verhindern / Gegenmaßnahmen ergreifen)
  
- Methoden:
  - Passive wie auch aktive Überwachung
  - von Systemen und Netzen / Netzsegmenten
  
- Im Folgenden:
  - Hostbasierte und netzbasierte IDS
  - Angriffserkennung

# Hostbasierte IDS (HIDS)

- Ausgeführt auf dem zu überwachenden System
  - Systemüberwachung
  - Applikationsüberwachung
  - Integritätsüberwachung (kryptographische Prüfsummen; Hashing)
- + Individuell an zu überwachendes System anpassbar
- + Sehr spezifische Aussagen über erkannte Angriffe möglich
- Benötigt Ressourcen des zu schützenden Systems
- HIDS selbst als Angriffsziel
- Anpassung an individuelles System erforderlich
- Fällt bei erfolgreichem Angriff mit angegriffenem System aus
- Bsp.:
  - Tripwire
  - AIDE (Advanced Intrusion Detection Environment)
  - Samhain

# Netz-basierte IDS (NIDS)

- Eigener Sensor (kein Gastsystem)
- Überwachen den Netzverkehr (Mithören):
  - eines Rechners
  - eines (Sub-) Netzes
  - einer gesamten Domäne
- + Ein Sensor für das gesamte Netz
- + NIDS kann „unsichtbar“ installiert werden
- + NIDS kann ausfallsicher installiert werden
- + Verteilte Angriffe erkennbar
- Betrieb am Mirror Port von Netzgeräten; Problem: Packet-Drop
- Überlast des gesamten NIDS möglich
- Verschlüsselte Daten und Kanäle
- „nur“ Netzauffälligkeiten erkennbar

# Angriffs- bzw. Mißbrauchserkennung

## ■ Signaturbasiert:

- Pattern Matching & Expertensysteme
- „typische Angriffsmuster“
- + Zuverlässigkeit
- Nur bekannte Angriffe erkennbar (keine Zero Day Exploits)
- Bsp.: Snort

## ■ Anomalie-Erkennung:

- Lernt „Normalverhalten“
- Abweichung wird als Angriff gedeutet
- + Flexibel bei neuen Angriffen
- Adaptivität bei Netzänderungen
- False Positives & False Negatives

## ■ Integritätsprüfung

- Berechnung kryptographischer Hashes
- Speicherung auf WORM und Vergleich
- + Schnell und Zuverlässig
- Aufwand bei gewollter Datenänderung
- Bsp.: Tripwire

## ■ Kombination der Verfahren

# Beispiel: Tripwire

- [www.tripwire.org](http://www.tripwire.org)
- Host-basiertes IDS
- Überwacht anzugebende Dateien und Verzeichnisse
- Erstellt (verschlüsselte) Datenbank mit Prüfsummen:
  - MD5
  - SHA-1
  - HAVAL, ...
- Berechnet regelmäßig Hashes und vergleicht mit gespeicherten
- Probleme:
  - Auswahl schützenswerter Objekte
  - Berechtigte Änderungen (z.B. Software-Updates)
  - Initialstatus muss sicher sein



# Beispiel: Snort

- [www.snort.org](http://www.snort.org)
- Netzbasiertes IDS
- Signatur-basierte Analyse
  - Regeln in Dateien definiert (\*.rules)
  - Alle Regeln mit logischem ODER verknüpft
  - Netzverkehr wird gegen diesen Regelsatz geprüft
- Konfiguration der Regeln
  - Manuell durch Administrator
  - Mitgelieferte Community Rules (ggf. Anpassungen erforderlich)
  - Kostenpflichtige, tagesaktuelle Vulnerability Research Team (VRT) Rules
    - z.B. Erkennen aktueller Exploits
    - fließen mit Verzögerung (30 Tage) in die Community Rules ein



# Snort-Konfiguration

## ■ Globale Variablen (snort.conf)

- Netzstruktur, üblicherweise
  - HOME\_NET (welche IP-Bereiche sind zu schützen?)
  - EXTERNAL\_NET (!HOME\_NET - alles andere)
- Bekannte Server („noisy machines“ im HOME\_NET), z.B.
  - DNS\_SERVERS
  - HTTP\_SERVERS

## ■ Präprozessoren, z.B.

- frag3 (zum Behandeln von in Fragmente zerlegte Angriffen)
- stream5 (zielspezifische stateful inspection von Kommunikationsabläufen, z.B. Reihenfolge von TCP SYN/FIN/Reset, Nutzdaten in SYN-Paketen (für einige OS in Ordnung), ...)

## ■ Output options, z.B.

- Logfiles
- Datenbanken (MySQL, PostgreSQL)

# Snort - Detection Engine Konfiguration

## ■ Generelles Vorgehen:

- ❑ Netzverkehr wird abgehört (sniffing)
- ❑ Pakete werden vorverarbeitet (preprocessing)
- ❑ Vergleich der Pakete mit in Rules angegebenen Parametern
- ❑ Entscheidung:
  - Verwerfen des Pakets (drop/reject; bei „inline“-mode)
  - Erzeugen eines Alarms (alert)
  - Paket protokollieren (log)
  - Ignorieren des Pakets (pass)

## ■ Aufbau von Rules: Header und Body

### ■ Header enthält:

- ❑ Action
- ❑ Protocol
- ❑ Source IP / Source Port(s)
- ❑ Operator
- ❑ Destination IP / Destination Port(s)

# Snort - Rules Body (1/2)

## ■ Body einer Rule enthält allgemein

- Event Message (Alarmmeldung, z.B. zum Finden in Logfiles)
- Patterns: Wann trifft die Regel auf ein Paket zu?

## ■ Mögliche Patterns

- content: Inhalt des Pakets
  - Als Sequenz von Hexzahlen oder ASCII (z.B. /bin/sh)
  - Groß-/Kleinschreibung wird ignoriert
  - depth (gibt Tiefe der Suche vor)
  - uricontent (Untersuchung des vom HTTP-Präprozessor extrahierten URI)
- flag
  - TCP-Flags (SYN, FIN, RST, ACK, URG, PSH)
  - Operatoren (+, \*, !) für Matching von Flag-Kombinationen

# Snort - Rules Body (2/2)

## ■ (Mögliche Patterns)

### □ flow

- to\_client, from\_server
- to\_server, from\_client
- established
- stateless

### □ threshold

- count n
- seconds m
- type
  - limit: Nur beim ersten Vorkommen von n Events pro Zeitintervall m,
  - threshold: Jedes n. Mal pro Zeitintervall m
  - both: Nur einmal pro Zeitintervall m nach n Events
- track (by\_src, by\_dst)

# Snort Rules - Beispiele

## ■ Beispiel Threshold:

Logge nur einen Event pro Minute, falls eigentlich mehr als 30 pro Minute auftreten

```
threshold: type both, track by_src, count 30, seconds 60
```

## ■ Beispielregel:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 22 (msg:"Internal SSH attack 60/60sec"; flags:S; threshold: type both, track by_src, count 60, seconds 60; sid:1000003;)
```

### **Header:**

Action: alert

Quelle: \$HOME\_NET, beliebiger Port

Operator: ->

Ziel: \$EXTERNAL\_NET, Port 22

### **Body:**

Event message: „Internal SSH...“

Flags: S = SYN

Logging: 1x pro Minute

falls mehr als 60 Events/Min

SID = Identifier der Rule