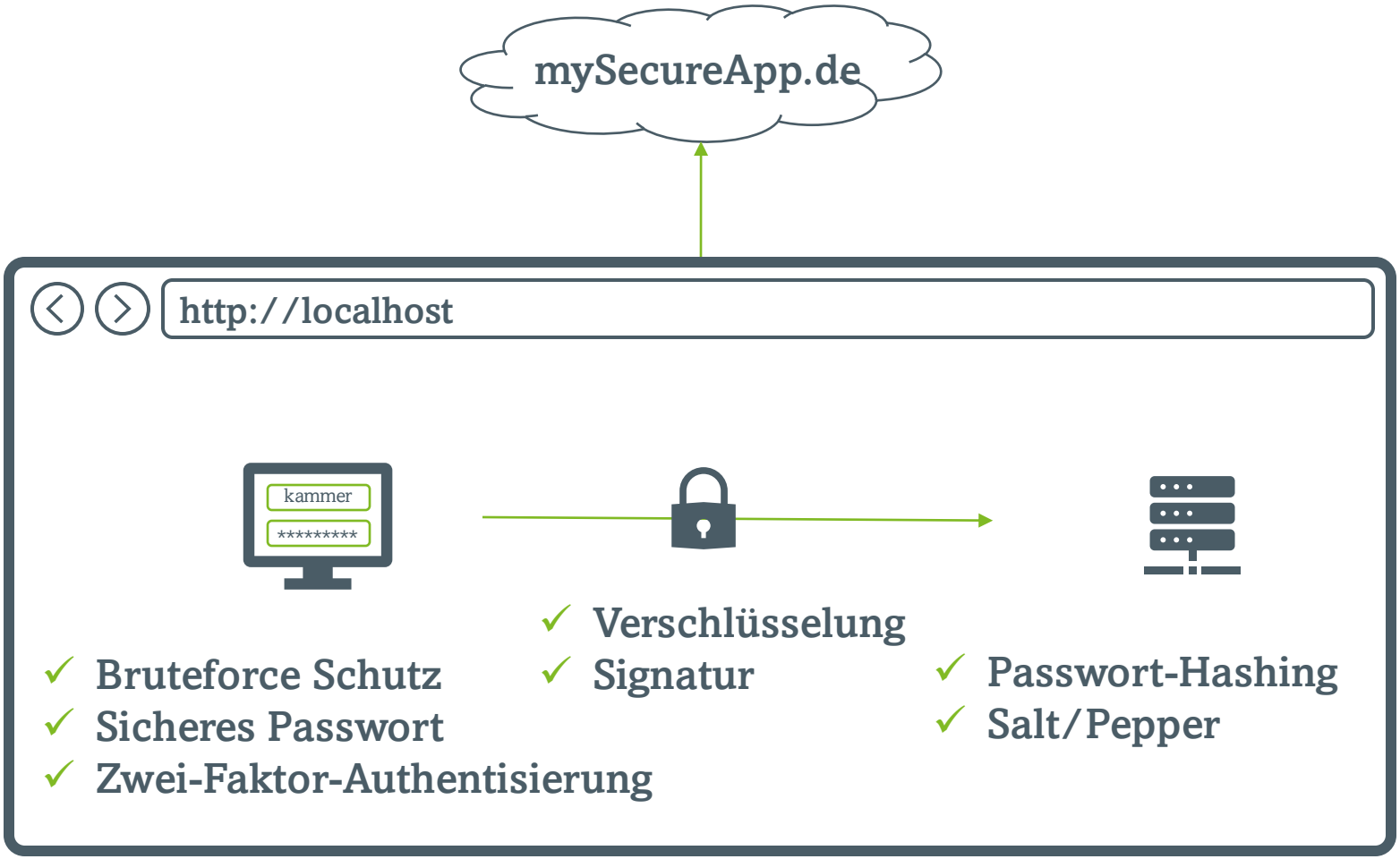
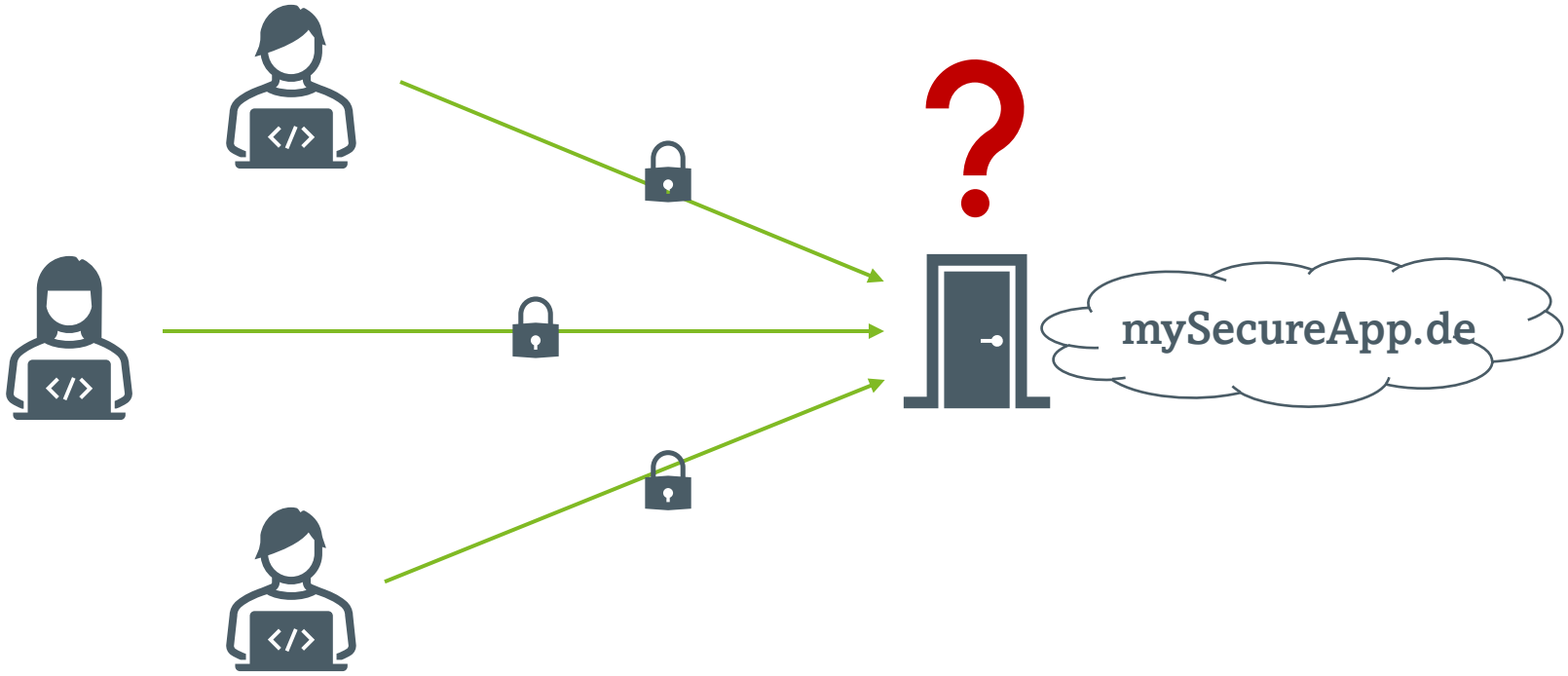
Authentifizierung und Autorisierung

Rückblick


mySecureApp.de
http://localhost
kammer
****
✓ Verschlüsselung
✓ Signatur
✓ Bruteforce Schutz
✓ Sicheres Passwort
✓ Zwei-Faktor-Authentisierung
✓ Passwort-Hashing
✓ Salt/Pepper

Rückblick



Begriffserklärung

Authentisierung

Benutzername

kammer

Passwort

****

Nutzer erbringt Identitätsnachweis

Authentifizierung

Prüfen des Identitätsnachweises

Autorisierung

Schreibrecht

Leserecht

Ausführungsrecht

Begriffserklärung

Authentisierung

Benutzername

kammer

- ✓ Sicheres Passwort
- ✓ Zwei-Faktor-Authentisierung

Nutzer erbringt Identitätsnachweis

Authentifizierung

Prüfen des Identitätsnachweises

Autorisierung

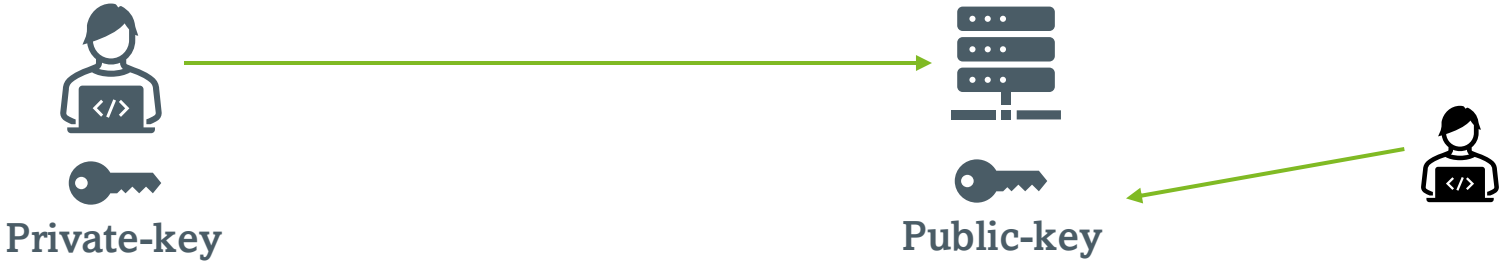
- Schreibrecht
- Leserecht
- Ausführungsrecht

Authentifizierung

- **Überprüfen des Identitätsnachweises**
  - Passwortüberprüfung
  - Zertifikatsüberprüfung
  - **Public-Key-Authentifizierung**

Public-Key-Authentifizierung

- Anmeldung eines Benutzers bei einem Server.



- Public-Key-Authentifizierung für Webanwendungen?

  JavaScript muss auf den privaten Schlüssel zugreifen können und das ist gefährlich für XSS-Attacken!

- Wird verwendet von SSH

Secure Shell (ssh)

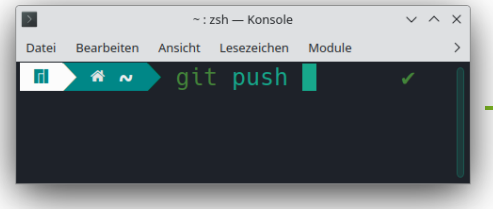
- Ziel: Aufbau einer verschlüsselten Netzwerkverbindung mit einem entfernten Gerät.
- Ersetzt telnet, rlogin, rsh, rcp und ftp (letzteres mithilfe von scp)
- Verschlüsselte Passwort- und Datenübertragung
- Direkte Unterstützung von verschlüsselter X11-Übertragung

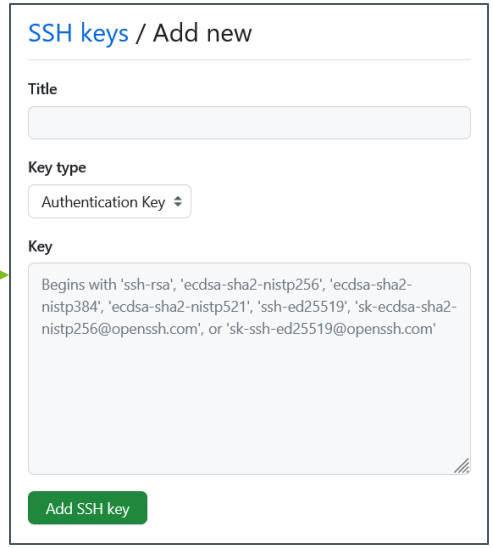
Secure Shell (ssh)

- **Authentisierung durch RSA**
  - Benutzer: Mit ssh-keygen privaten/öffentlichen Schlüssel generieren
  - Rechner: Verifikation des Remote-Rechners
- **Verschlüsselung durch verschiedene (wählbare) Methoden:**
  - AES, Blowfish, 3DES, ...
- **OpenSSH gibt es für Linux, Windows und MAC.**

Secure Shell (ssh)

GitHub





ssh - Remote Verbindung aufbauen

ssh <Benutzername>@<entfernterRechner> [<Kommando>]

Beispiel:

ssh kammer@fk-vv.mni.thm.de

ssh - Datei auf Remote-Rechner kopieren

scp file name@remote:destination

Beispiel:

scp ./test.txt kammer@ fk-vv.mni.thm.de :~/tmp

ssh - Asymmetrisches Schlüsselpaar erstellen

Generiert privaten+öffentlichen Schlüssel im Verzeichnis ~/.ssh

ssh-keygen -t rsa

ssh-keygen -t dsa

ssh-keygen -t rsa1

Identifikation mit Schlüsseln (Keine Passwortabfrage mehr!!)

Anhängen des Inhaltes aus localhost:~/.ssh/*.pub

in die Datei remotehost:~/.ssh/authorized_keys

ssh - Server

- **Installation**
  - ssh-Daemon installieren und starten
  - Firewall Port 22/tcp öffnen
  - Per ssh auf das System verbinden.
  - Fertig?

Hacker können beliebig lange Brute-Force Attacken durchführen.

ssh - Server

- **Oft erfolgen diese auf den Nutzer root.**
- **Mögliche Lösungen:**
  - ssh nur für Bestimmte Benutzergruppen erlauben

    /etc/ssh/sshd_config
    PermitRootLogin no
    AllowGroups sshusers
    (Passende User der Gruppe hinzufügen)

ssh - Bruteforce ist „Internet Noise“

Apr 13 22:57:12 sshd[167302]: Failed password for invalid user test from 201.236.xxx.xxx port 56706 ssh2
Apr 13 22:57:16 sshd[168302]: Failed password for root from 49.88.xxx.xxx port 43324 ssh2
Apr 13 22:57:20 sshd[168302]: Failed password for root from 49.88.xxx.xxx port 43324 ssh2
Apr 13 22:57:23 sshd[168302]: Failed password for root from 49.88.xxx.xxx port 43324 ssh2
Apr 13 22:57:23 sshd[169780]: Invalid user emilio from 115.66.xxx.xxx port 59044
Apr 13 22:57:34 sshd[175112]: Invalid user probe from 113.98.xxx.xxx port 46929
Apr 13 22:57:36 sshd[173807]: Invalid user recording from 119.96.xxx.xxx port 59454
Apr 13 22:57:38 sshd[178130]: Invalid user isp from 49.234.xxx.xxx port 38984
Apr 13 22:57:54 sshd[178183]: Failed password for root from 139.59.xxx.xxx port 45732 ssh2
Apr 13 22:58:09 sshd[178198]: Invalid user admin from 177.8.xxx.xxx port 45131
Apr 13 22:58:11 sshd[178198]: Failed password for invalid user admin from 177.8.xxx.xxx port 45131 ssh2
Apr 13 22:58:41 sshd[178210]: Failed password for root from 80.246.xxx.xxx port 60502 ssh2
Apr 13 22:58:44 sshd[178214]: Invalid user chay from 58.64.xxx.xxx port 45652
Apr 13 22:58:45 sshd[178216]: Invalid user oracles from 115.66.xxx.xxx port 50

Authentifizierung Webanwendung

- HTTP ist zustandslos
- Wo liegt darin das Problem?
  - Der Nutzer muss sich bei jeder Anfrage neu authentisieren.

Authentifizierung HTTP(S)

- **Welche Möglichkeiten haben wir?**
  - Cookie-based Authentifizierung
  - Authentifizierung per HTTP-Header
  - Cookieless Authentifizierung

Speicherung im Browser

Local Storage

- Key/Value Speicher
- Erlaubt eine große Datenmenge
- Persistent
- Zugriff nur von der Website

Session Storage

- Key/Value Speicher
- Erlaubt eine große Datenmenge
- Speicher entfernt nach schließen des Browsers
- Zugriff nur von der Website und dem Tab

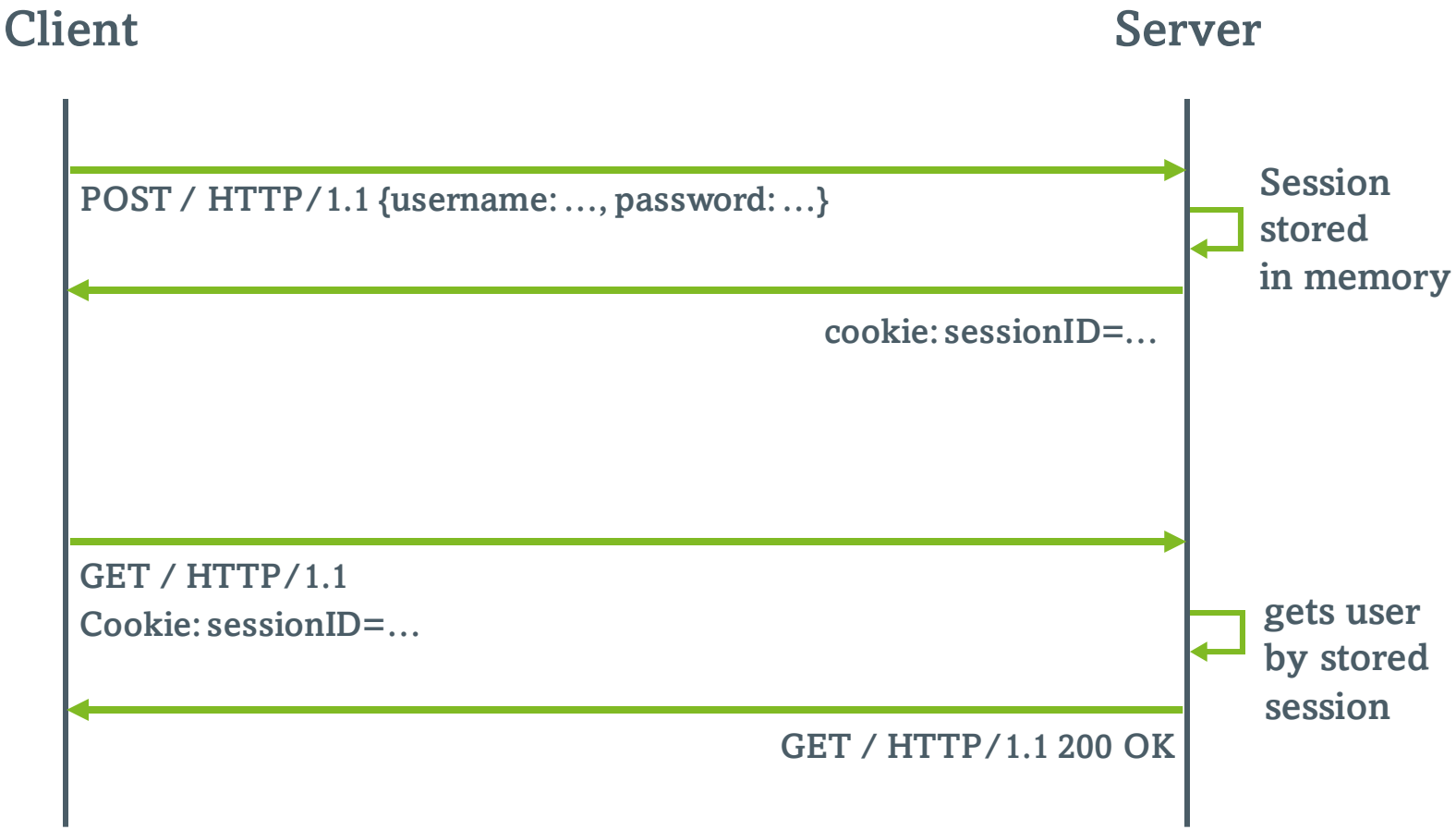
Cookies

- Name, Version, Ablaufdatum, …
- Maximal 4KB
- Persistent und Session Cookies
- Zugriff nur von der Website

Cookies

- Können durch den Server und durch den Client gesetzt werden.
- Aufbau (Auszug):
  - Name
  - Wert
  - Domain
  - Ablaufdatum
  - HttpOnly
  - Secure
- HttpOnly verhindert, dass ein Client diesen Cookie auslesen kann.
- Cookies mit dem Attribut *secure* werden nicht über HTTP versendet.

Authentifizierung durch Cookies



Authentifizierung per HTTP-Header

Headers

- WWW-Authenticate
- Proxy-Authenticate
- Authorization
- Proxy-Authorization

Authentication schemes

- Basic (RFC 7617)
- Bearer (RFC 6750)
- Digest (RFC 7616)
- HOBA (RFC 7486)
- Mutual (RFC 8120)
- Negotiate / NTLM (RFC4599)
- VAPID (RFC 8292)
- SCRAM (RFC 7804)
- AWS4-HMAC-SHA256

Basic Auth

- Authentisieren bei einem Webserver mittels Benutzername und Passwort
- Benutzername und Passwort werden base64 encodiert

```
Basic <Base64 encoded username and password>
```

Basic Auth


Client
Server
GET / HTTP/1.1
GET / HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm=„Access to the staging site“
Ask user
GET / HTTP/1.1
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
Check credentials
GET / HTTP/1.1 200 OK
or
GET / HTTP/1.1 401 Unauthorized

Sicherheitsprobleme

- Wird bei jedem Request an den Server gesendet
- Benutzername und Passwort nur encodiert nicht verschlüsselt!

> HTTPS nutzen um eine Verschlüsselte Verbindung aufzubauen

Bearer Token

- Autorisieren von Anfragen mittels eines Zugriffsschlüssels
- Wird im Header der Anfrage übergeben
- Nicht an eine Identität gebunden
- Nur über HTTPS sicher!

Bearer Token

JWT Token

- Nutzerinformationen werden im Token gespeichert

Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload

```
{
  "sub": "1234567890",
  "name": "Frank Kammer",
  "admin": true
}
```

Signatur

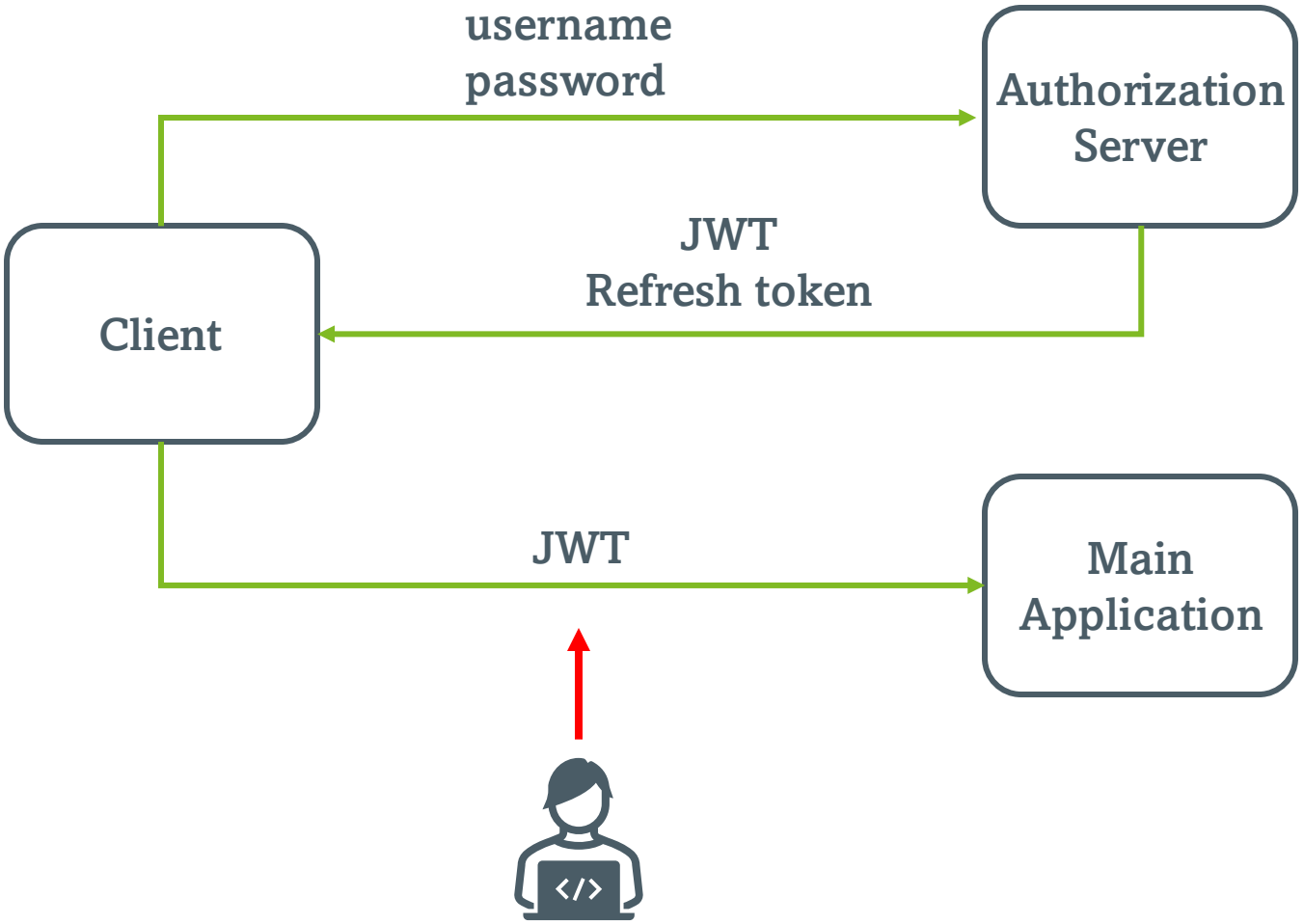
```
HMACSHA256(
base64UrlEncode(header) +
"." +
base64UrlEncode(payload),
secret);
```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkZyYW5rIEthbW1lciIsImFkbWluIjp0cnVlfQ.0l7YwSlgWJJIRV985fQOLH2MZOLfvBt3shzguPGz94

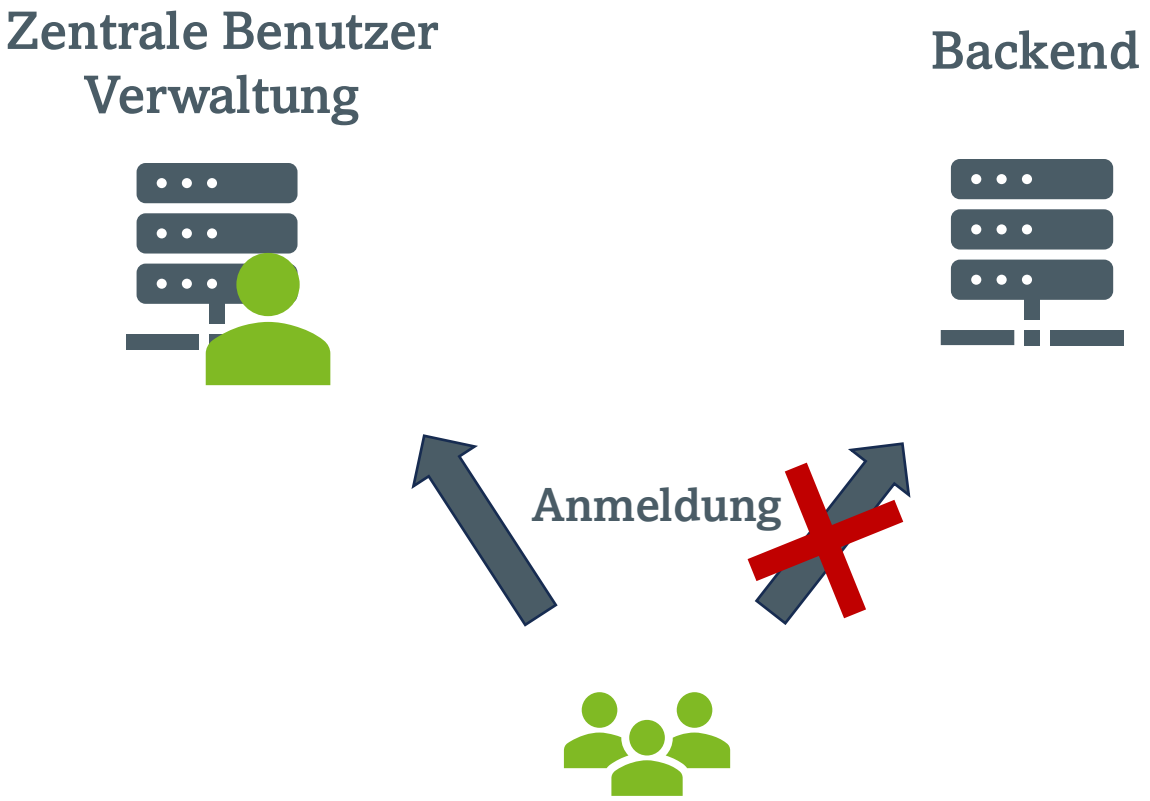
Refresh Token

- JWT Token läuft nach einer bestimmten Zeit ab
- Wir wollen uns nicht immer erneut anmelden

Refresh Token



Single-Sign-On



Single-Sign-On

Single-Sign-On Vorteile

- Anmeldung innerhalb einer Organisation nur 1x notwendig.
- Nutzer können zentral verwaltet werden.
- Nur eine Implementierung notwendig.

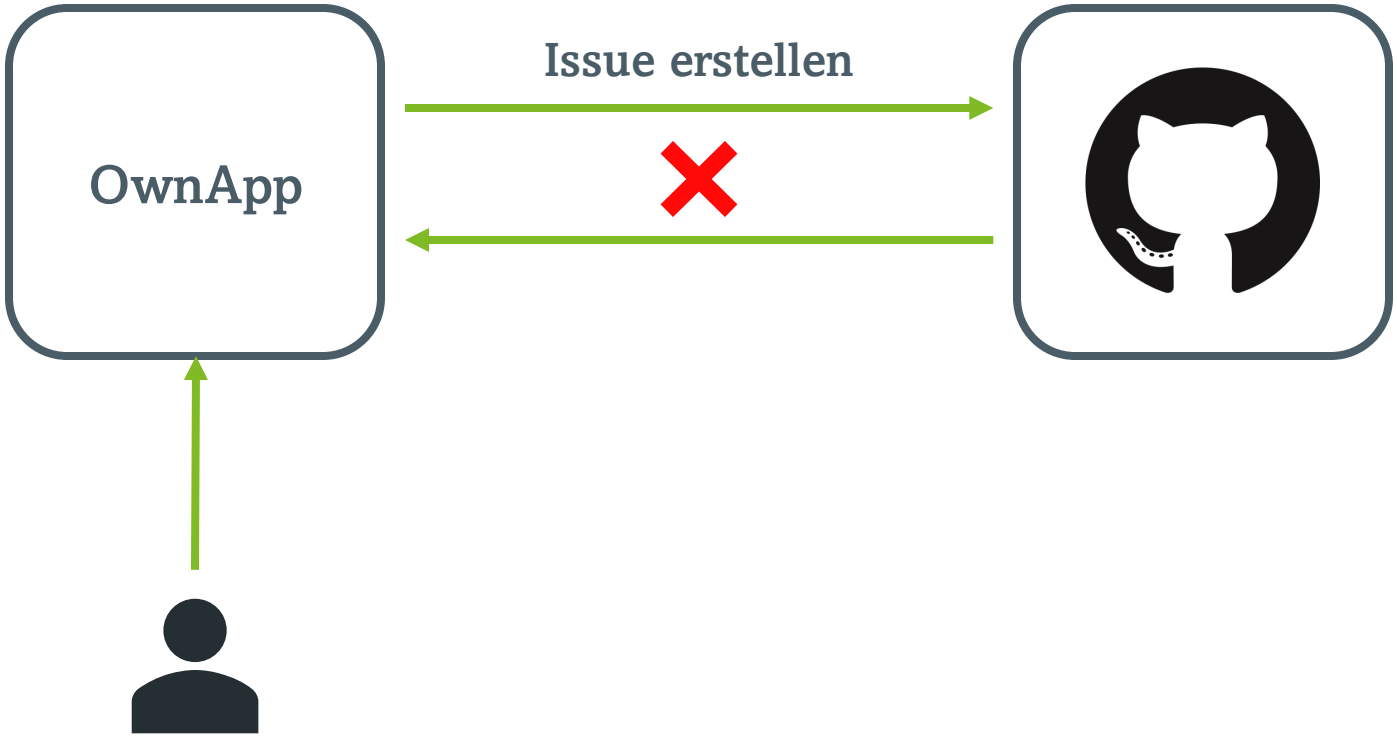
Single-Sign-On Protokolle

- OpenID-Connect (OAuth)
- CAS
- Shibboleth
- Security Assertion Markup Language (SAML)

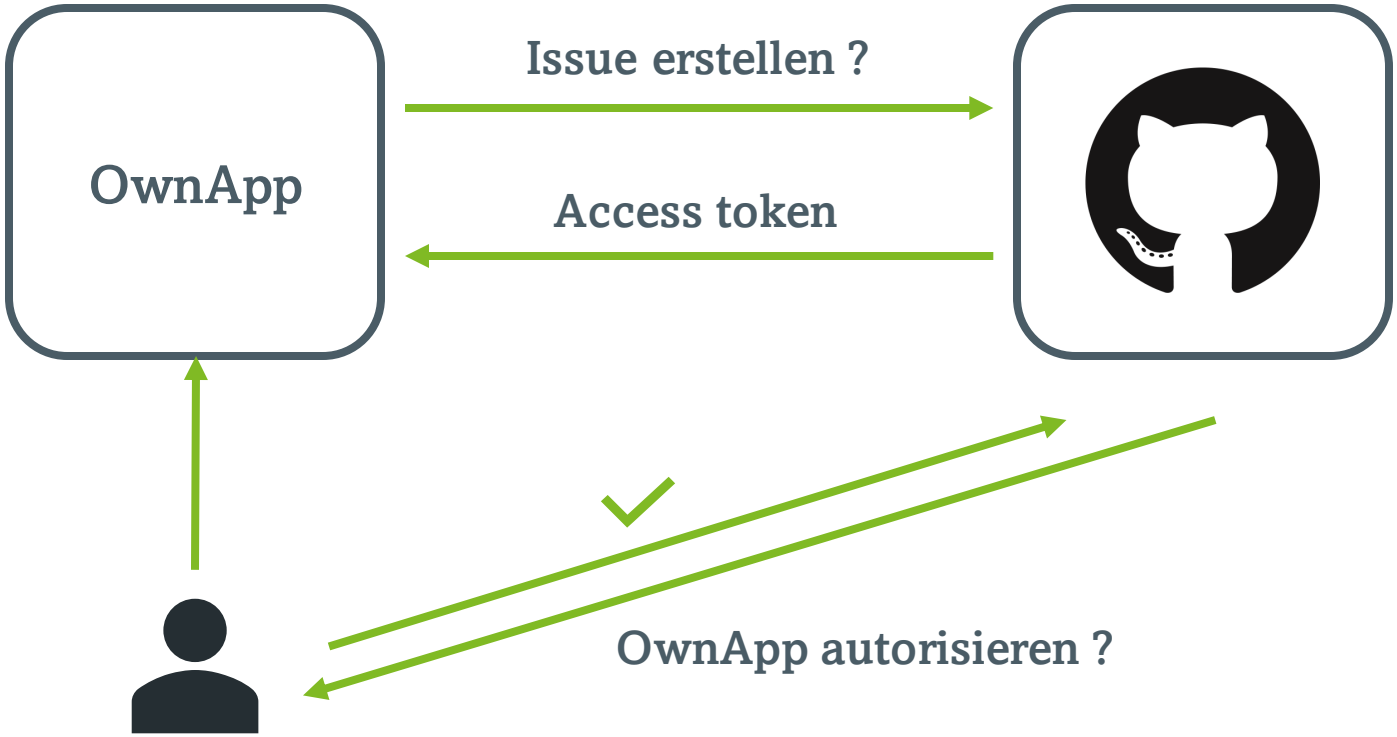
OAuth

- Open-Authorization-Protocol
- O-Auth 1.0 und O-Auth 2.0
- Autorisierung nicht Authentifizierung!
- Ursprünglich für die Autorisierung zwischen Services
- Für die Authentifizierung z.B. OpenID Connect

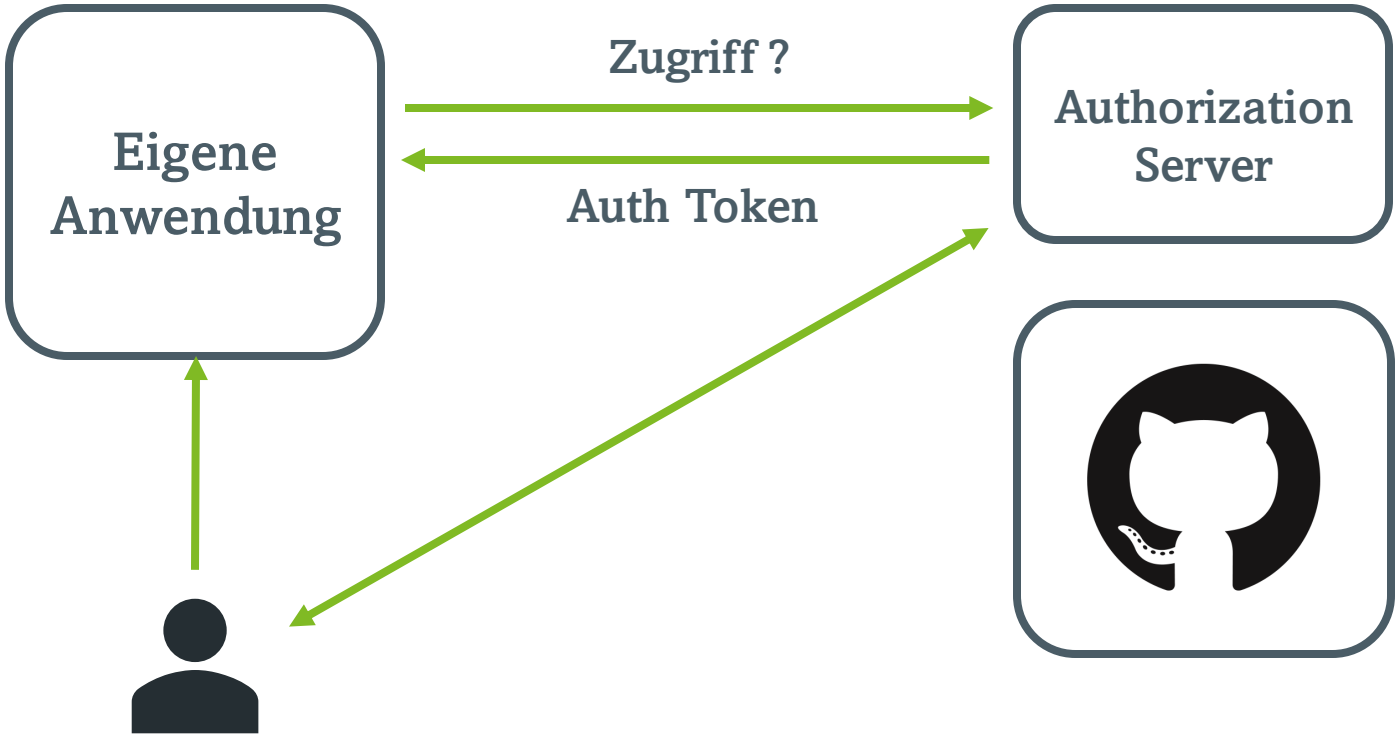
OAuth 2.0



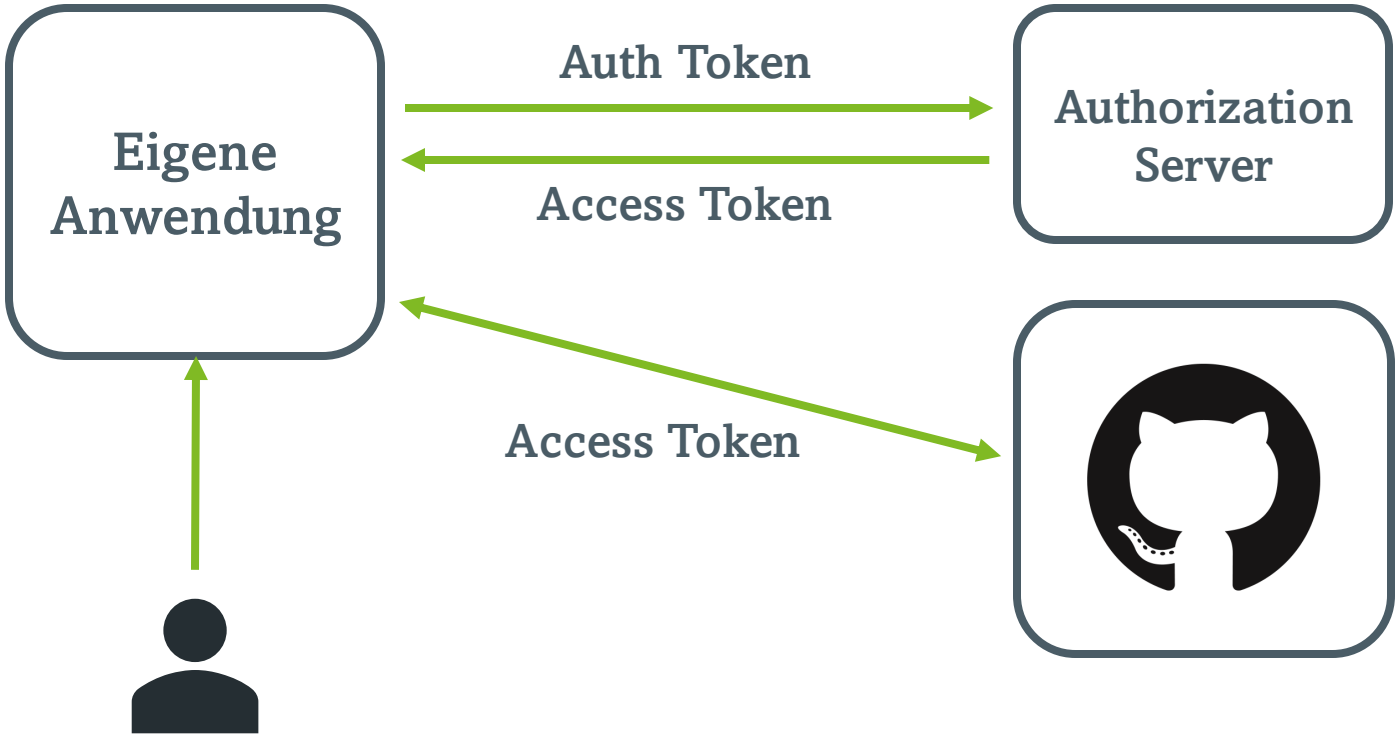
OAuth 2.0



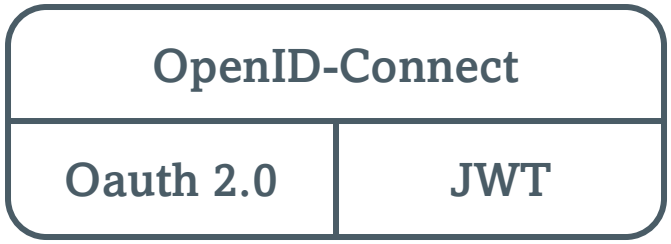
OAuth 2.0


Eigene Anwendung
Zugriff ?
Auth Token
Authorization Server

OAuth 2.0

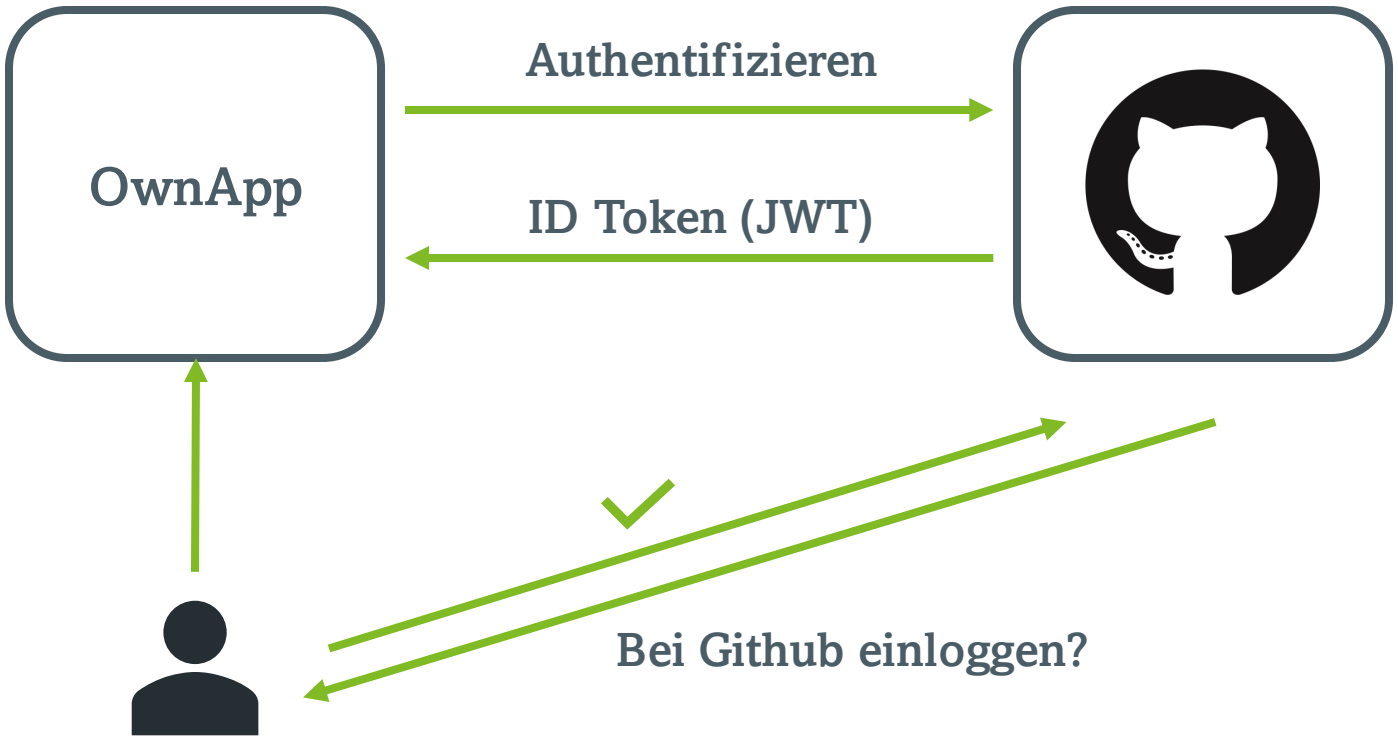

Eigene Anwendung
Auth Token
Access Token
Authorization Server
Access Token

OpenID-Connect

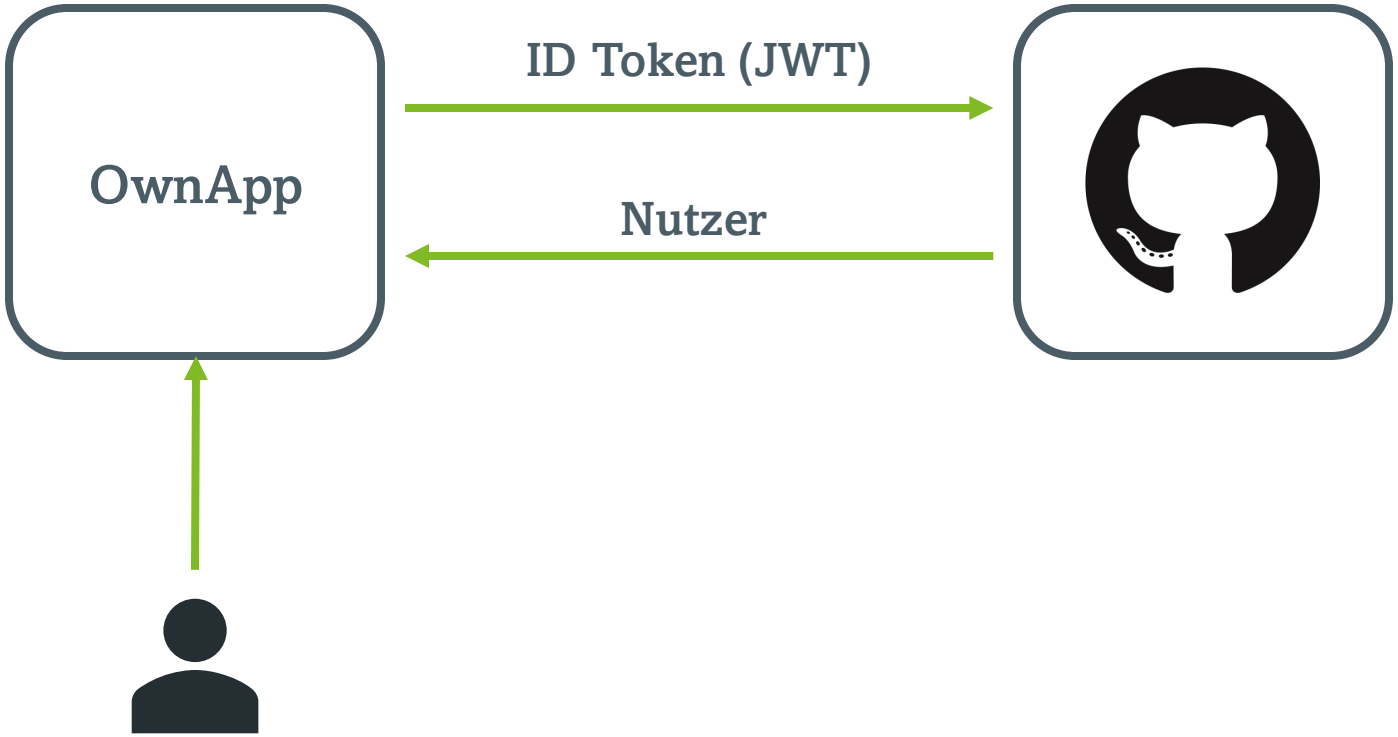


- Liegt über dem OAuth Protokoll und erweitert dies um einen Authentifizierung des Nutzers

OpenID-Connect



OpenID-Connect



Mögliche Fehler - Single-Sign-On

- Leere Zeichenkette als Password

Leere Zeichenkette als Password

Anlegen des Externen Benutzers bei der Anmeldung

Zentrale Benutzer Verwaltung

```
{
  "uid": "test",
  "name": "Test User",
  "password": ""
}
```

Lokale Benutzer

Leere Zeichenkette als Password

Anlegen des Externen Benutzers bei der Anmeldung

Zentrale Benutzer Verwaltung

```
{
  "uid": "test",
  "name": "Test User",
  "password": ""
}
```

Lokale Benutzer

Anmeldung mit leerer Zeichenkette möglich

Leere Zeichenkette als Password

Anlegen des Externen Benutzers bei der Anmeldung

Zentrale Benutzer Verwaltung

```
{
  "uid": "test",
  "name": "Test User",
  "password": null
}
```

Lokale Benutzer

Besser: `null` oder zufälliges Password verwenden

Mögliche Fehler - Single-Sign-On

- Zuordnung von Benutzern

Authentifizierung Webanwendung

- **Der Nutzer muss sich bei jeder Anfrage neu authentisieren.**

- In bestimmten Fällen möchte man das, z.B., um eine sensible Operation durchzuführen.

Begriffserklärung

Authentisierung

Benutzername

kammer

- ✓ Sicheres Passwort
- ✓ Zwei-Faktor-Authentisierung

Nutzer erbringt Identitätsnachweis

Authentifizierung

- ✓ Passwortüberpr.
- ✓ Zertifikatsüberpr.
- ✓ ...

Prüfen des Identitätsnachweises

Autorisierung

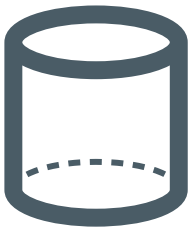
Schreibrecht

Leserecht

Ausführungsrecht

Zugriffskontrolle – Grundbegriffe

- **Subjekt / Akteur (subject): Initiator der Handlung**
  - Benutzer (z.B. Alice, Bob), Gruppen, Prozesse, Rechner/Geräte

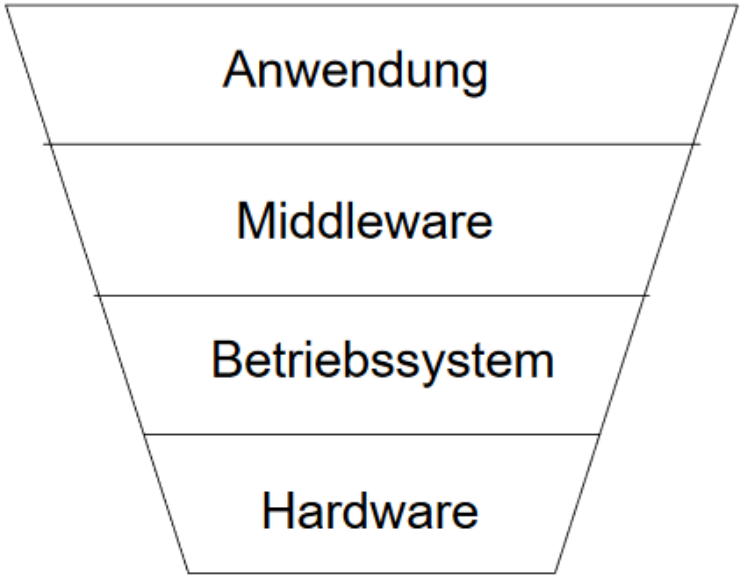


- **Objekt (object): Gegenstand der Handlung**
  - Ressource Dateien, SQL-Tabellen, Konto, Prozesse
  - Achtung: Subjekte können auch Objekte sein

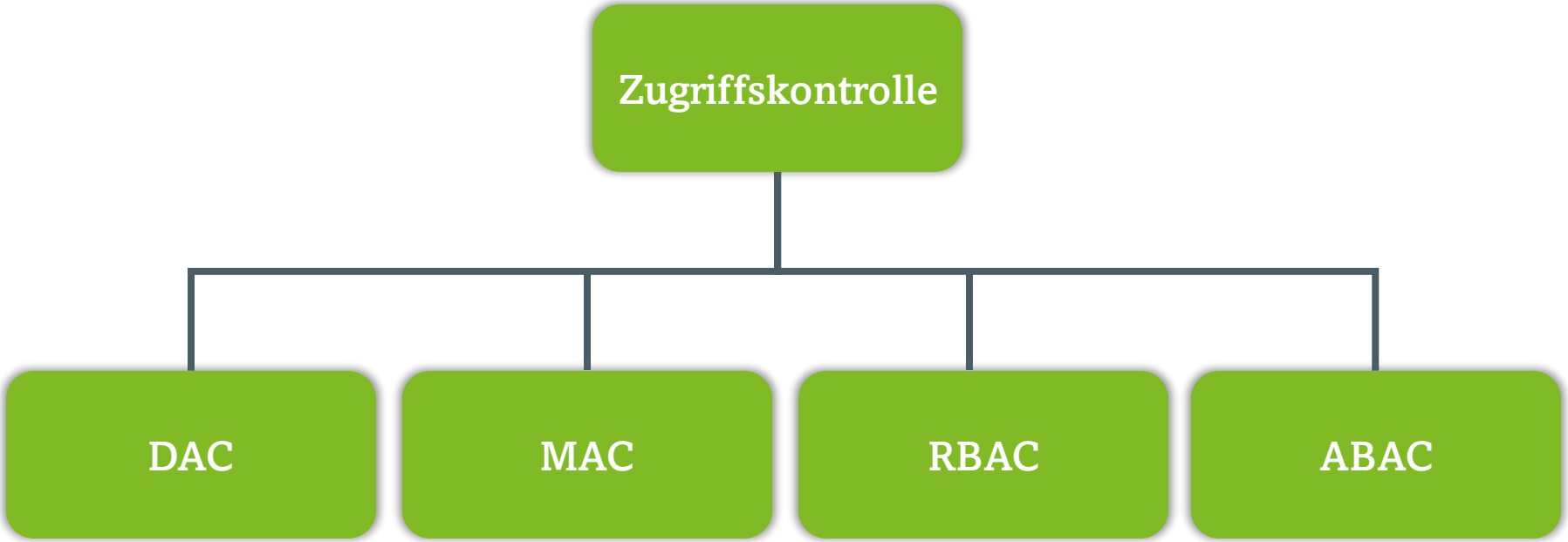
- **Berechtigung (permission) bzw. Zugriffsrecht (access right):**
  - Was genau darf gemacht werden?
  - Z.B.: lesen, schreiben, ausführen, löschen

Ebenen der Zugriffskontrolle

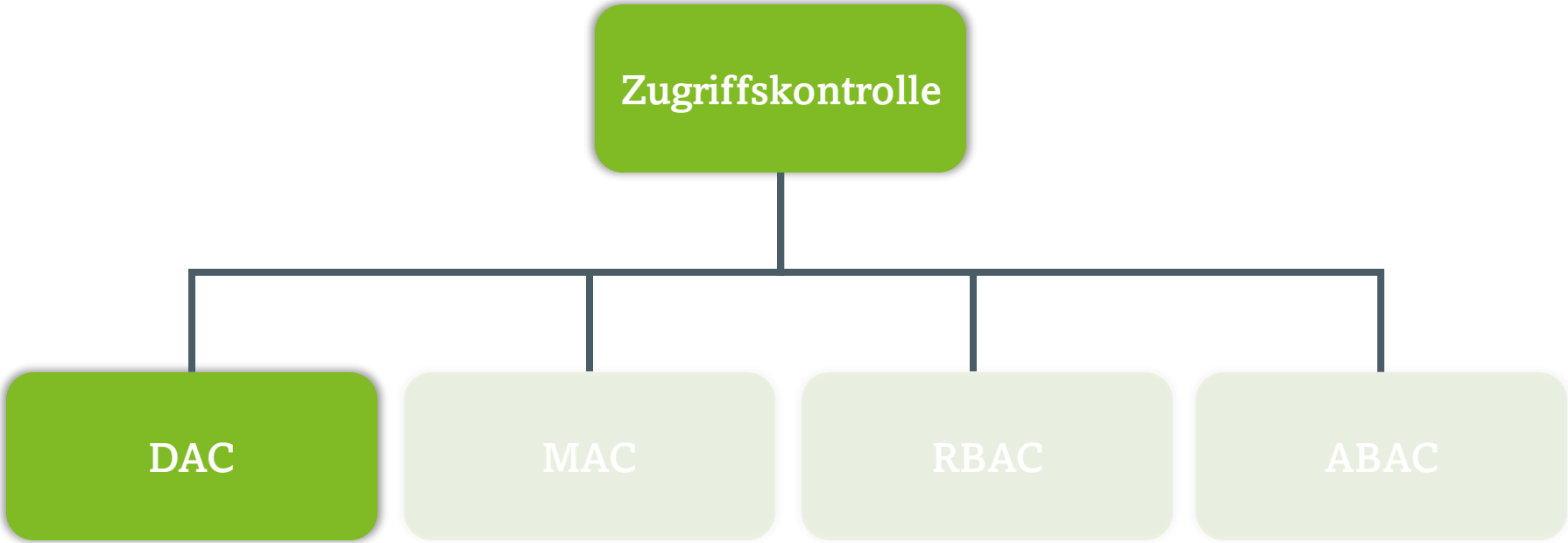
- Abhängigkeit der höheren Ebenen von den Zugriffskontrollmechanismen der niedrigeren Ebenen
- Zugriffskontrollmechanismen sind in höheren Ebenen komplexer
- Fehler in niedrigeren Ebenen können zum Umgehen der Sicherheitsmechanismen der höheren Ebenen führen



Arten der Zugriffskontrollen

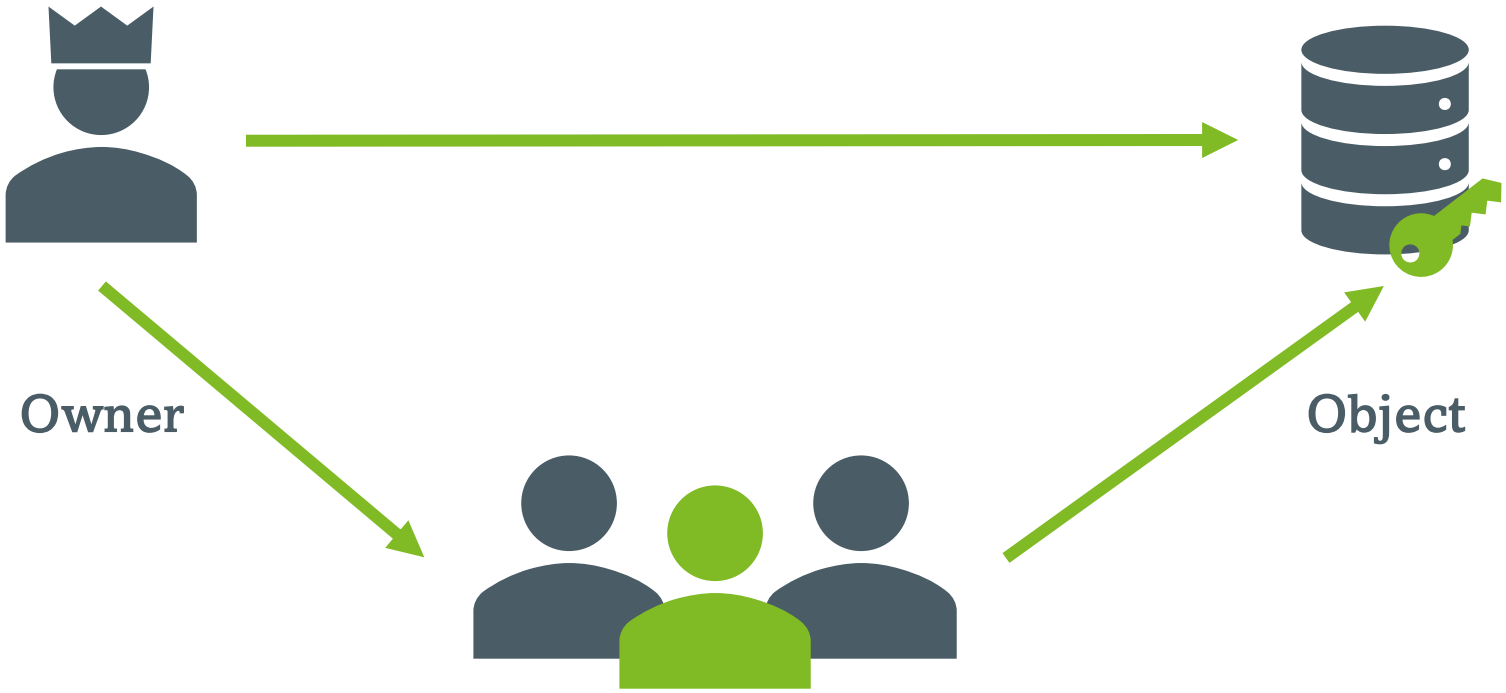


Arten der Zugriffskontrollen



Discretionary Access Control (DAC)

- Nutzer entscheiden über die Zugriffsberechtigungen



Discretionary Access Control (DAC)

- Nutzer entscheiden über die Zugriffsberechtigungen

Zugriffsmatrix (Lampson 1974):

Zugriffskontrolle wird in ihrer grundlegenden Form dargestellt durch eine Zugriffskontrollmatrix (access control matrix) M mit:

$$M: S \times O \rightarrow 2^R$$

wobei S die Menge der Subjekte, O die Menge der Objekte und R die Menge der Zugriffsrechte (wie z.B. read, write, execute) sind

Zugriffsmatrix

Subjekt/ Objekt	Datei 1	Datei 2	Datei 3	Prozess 1
Alice	{read, write}		{write}	
Bob		{execute}		{suspend}
Janet		{read}	{read}	
Frank	{read}			

Zugriffsmatrix

- **Zugriffsmatrix ist oft dünn besetzt**
  - d.h. eine vollständige Speicherung ist zu speicherintensiv O(|S| * |O|)

> **Lösung: Zugriffskontrolllisten**

Zugriffskontrolllisten (ACLs)

- Subjekte und deren Zugriffsrechte werden beim Objekt gespeichert
- Realisierung der Matrix nach Spalten

Zugriffsmatrix

Subjekt/ Objekt	Datei 1	Datei 2	Datei 3	Prozess 1
Alice	{read, write}		{write}	
Bob		{execute}		{suspend}
Janet		{read}	{read}	
Frank	{read}			

Zugriffskontrolllisten (ACLs)

- Subjekte und deren Zugriffsrechte werden beim Objekt gespeichert
- Realisierung der Matrix nach Spalten
- Beispiele:
  - ACL(Datei1) = ((Alice, {read, write}),(Frank, {read}))
  - ACL(Datei2) = ((Bob, {execute}), (Janet, {read}))

> Zugriffskontrolllisten sind das am häufigsten eingesetztes Konzept für die Zugriffskontrolle

Zugriffskontrolllisten (ACLs)

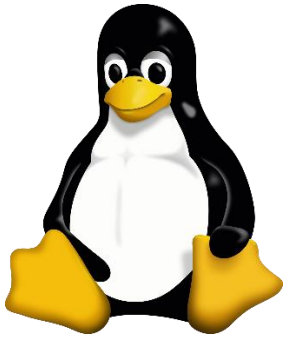
Vorteile:

- Vergebene Zugriffsrechte sind effizient für Objekte bestimmbar
- Rechterücknahme (pro Objekt) ist meist effizient realisierbar
- Einfach zu implementieren

Nachteile:

- Bestimmen der Subjekt-Rechte i.d.R. sehr aufwendig
- Aufwendige ACL-Kontrollen bei jedem Zugriff
- Schlechte Skalierbarkeit bei sehr dynamisch wechselnden Nutzern (Subjekten), falls ACLs für einzelne Nutzer vergeben werden

Zugriffskontrolllisten: Beispiel



Zugriffskontrolle in Linux

Zugriffskontrolle in Linux

Subjekte:

- Benutzer: User-ID, Gruppen-ID, Hilfsgruppen IDs (bis zu 16)
- Prozesse: User-ID/Group-ID (effective, real, saved)

Dateien:

- Owner (User-ID), Group (Group-ID)
- rwx-bits für: Owner, Group, Other
- sUid (Suid-Bit), sGid: Rechteveränderung bei Dateiausführung

Verzeichnisse:

- t-Bit (nur der Eigentümer darf Dateien aus Verzeichnis löschen)
- sGid-Bit (steuert Gruppe bei Datei-Erzeugung)

Zugriffskontrolle in Linux

linuxpc:~> ls –l

drwx--x—x	2 frank users	4096	27. Mär 12:02 bin
-rw-r--r--	1 frank users	8642048	22. Mär 14:07 SSE_Teil01+2.ppt
-rwxr--r--	1 frank users	14766080	22. Mär 14:07 SSE_Teil03.ppt
-rw-r--r--	1 frank users	899584	24. Mär 00:35 SSE_Teil04.ppt

Zugriffskontrolle in Linux

linuxpc:~> ls –l

```
drwx--x--x    2 frank users    4096     27. Mär 12:02 bin
-rw-r--r--    1 frank users    8642048  22. Mär 14:07 SSE_Teil01+2.ppt
-rwxr--r--    1 frank users    14766080 22. Mär 14:07 SSE_Teil03.ppt
-rw-r--r--    1 frank users    899584   24. Mär 00:35 SSE_Teil04.ppt
```

Dateityp

Zugriffskontrolle in Linux

linuxpc:~> ls –l

drwx--x--x	2 frank users	4096	27. Mär 12:02 bin
-rw-r--r--	1 frank users	8642048	22. Mär 14:07 SSE_Teil01+2.ppt
-rwxr--r--	1 frank users	14766080	22. Mär 14:07 SSE_Teil03.ppt
-rw-r--r--	1 frank users	899584	24. Mär 00:35 SSE_Teil04.ppt

User (Owner)

Zugriffskontrolle in Linux

```
linuxpc:~> ls –l

drwx--x--x    2 frank users    4096      27. Mär 12:02 bin
-rw-r--r--    1 frank users    8642048   22. Mär 14:07 SSE_Teil01+2.ppt
-rwxr--r--    1 frank users    14766080  22. Mär 14:07 SSE_Teil03.ppt
-rw-r--r--    1 frank users    899584    24. Mär 00:35 SSE_Teil04.ppt
```

Group

Zugriffskontrolle in Linux

linuxpc:~> ls –l

drwx--x--x	2 frank users	4096	27. Mär 12:02 bin
-rw-r--r--	1 frank users	8642048	22. Mär 14:07 SSE_Teil01+2.ppt
-rwxr--r--	1 frank users	14766080	22. Mär 14:07 SSE_Teil03.ppt
-rw-r--r--	1 frank users	899584	24. Mär 00:35 SSE_Teil04.ppt

Other

Zugriffskontrolle in Linux

linuxpc:~> ls –l

drwx--x--x	2 frank users	4096	27. Mär 12:02 bin
-rw-r--r--	1 frank users	8642048	22. Mär 14:07 SSE_Teil01+2.ppt
-rwxr--r--	1 frank users	14766080	22. Mär 14:07 SSE_Teil03.ppt
-rw-r--r--	1 frank users	899584	24. Mär 00:35 SSE_Teil04.ppt

r = read

w = write

x = execute

Zugriffskontrolle in Linux

```
linuxpc:~> ls -l|grep ^...x

drwx--x--x    2 frank users 4096        27. Mär 12:02 bin
-rwxr--r--    1 frank users 14766080    22. Mär 14:07 SSE_Teil03.ppt
```

Zugriffskontrolle in Linux

Besitzer und Rechte ändern:

linuxpc:~> chmod 777 SSE_Teil01+2.ppt

linuxpc:~> sudo chown alex:users *

linuxpc:~> ls -l
drwx--x--x 2 alex users 4096 27. Mär 12:02 bin
-rwxrwxrwx 1 alex users 8642048 22. Mär 14:07 SSE_Teil01+2.ppt
-rwxr--r-- 1 alex users 14766080 22. Mär 14:07 SSE_Teil03.ppt
-rw-r--r-- 1 alex users 89958 24. Mär 00:35 SSE_Teil04.ppt
-rw-r----- 1 alex users 89958 24. Mär 00:35 test

Zugriffskontrolle in Linux

- 1 = x (eXecute)
- 2 = w (Write)
- 3 = wx (Write/eXecute) = 1 + 2
- 4 = r (Read)
- 5 = rx (Read/eXecute) = 4 + 1
- 6 = rw (Read/Write) = 4 + 2
- 7 = (Read/Write/eXecute) = 4 + 2 + 1

Zugriffskontrolle in Linux

Standardrechte von neuen Dateien:

linuxpc:~> umask 137

linuxpc:~> touch test

linuxpx:~> ls -l
---x-wxrwx 1 frank frank 0 27. Mär 12:02 test

Zugriffskontrolle in Linux

Schutzbits sind einfache Form von ACL

Informationen für die Zugriffskontrolle den Dateien zugeordnet

Bislang (nicht) möglich

- Alice kann nicht Bob allein Lese-Rechte auf eine bestimmte Datei von Ihr geben.
- Benutzung von Gruppen hilft etwas: Root kann Gruppen anlegen.

Zusätzliches ACL-Konzept: getfacl, setfacl

getfacl, setfacl

Es gibt zwei grundlegende Klassen von ACLs:

Minimale ACL:

Einträge für die Typen "owner", "owning group" und "other„

Erweiterte ACL (Eintrag mask existiert):

- Einträge "owner", "owning group" und "other"
- Zusätzlich keine/beliebig viele Einträge der Typen
  - "named user„
  - "named group“

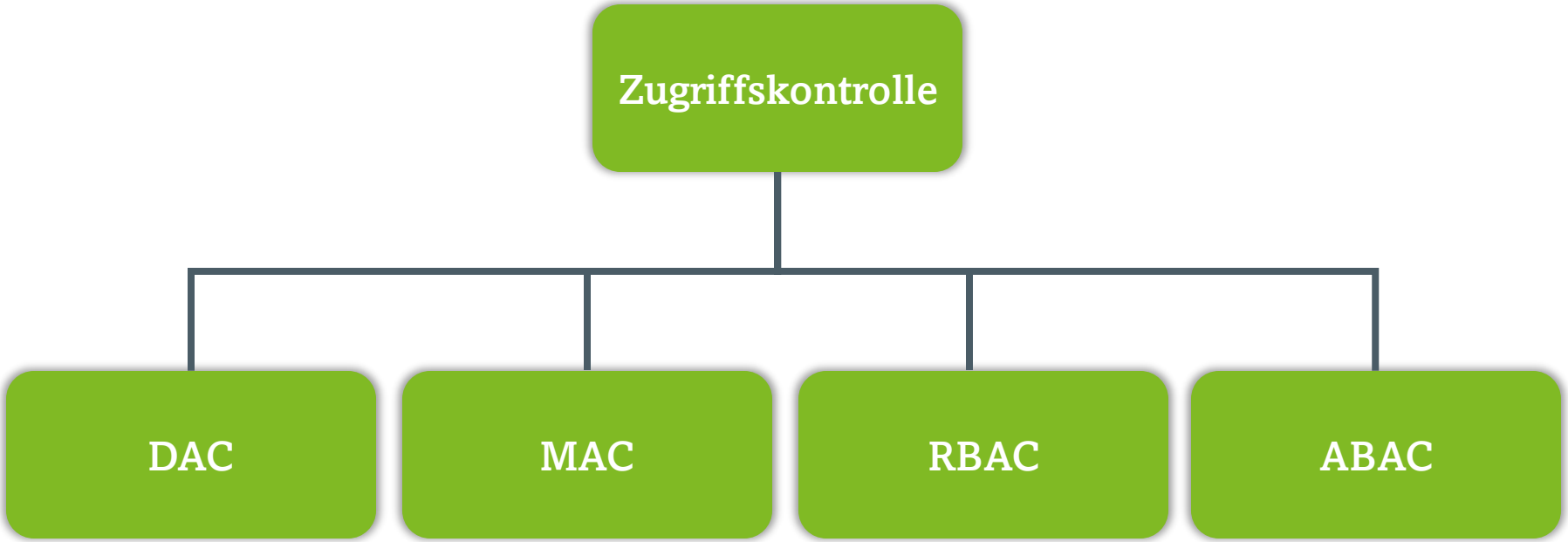
getfacl, setfacl

Typ	Textformat
owner	user::rwx
named user	user:name:rwx
owning group	group::rwx
named group	group:name:rwx
mask	mask::rwx
other	other::rwx

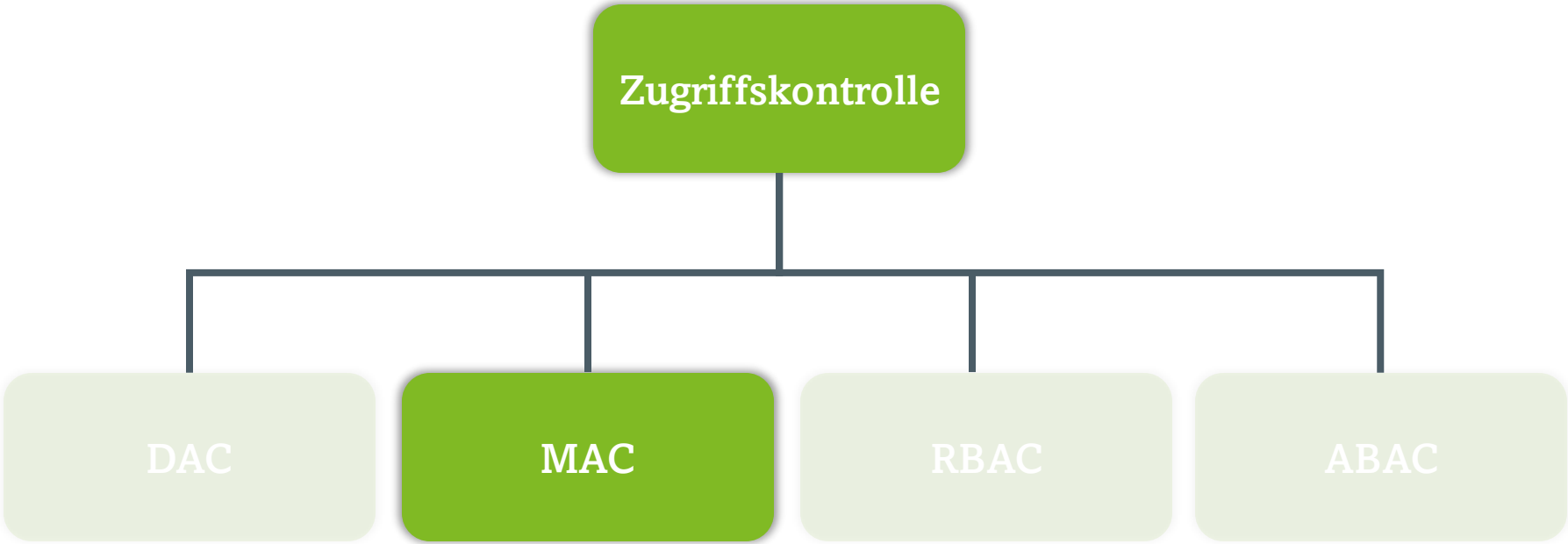
getfacl, setfacl

```
linuxpc:~> ls –l bericht0501
-rw-r----- 1 her versand  188 Jul 12 9:13 bericht0501

linuxpc:~> setfacl –m user:doo:6,group:einkauf:4 bericht0501
-rw-r-----+ 1 her versand  188 Jul 12 9:13 bericht0501

linuxpc:~> getfacl bericht0501
# file: bericht0501
# owner: her
# group: versand
User::rw-
User:doo:rw- #effective:r--
```

Arten der Zugriffskontrollen

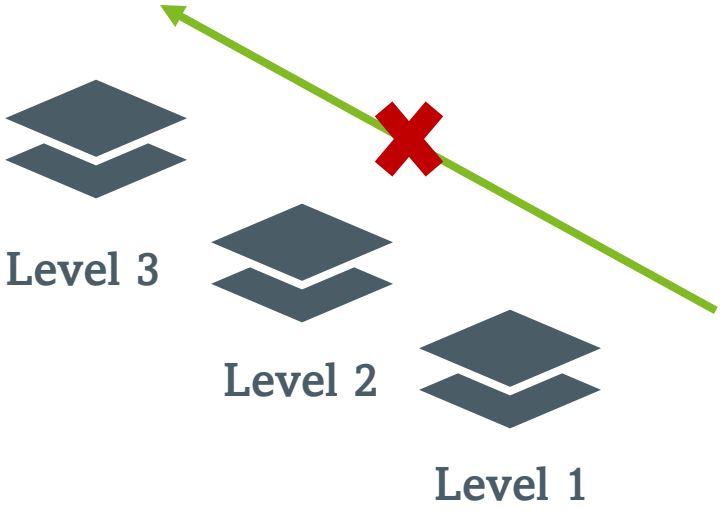


Arten der Zugriffskontrollen



Mandatory Access Control (MAC)

- Entscheidung über Zugriffsberechtigung auf Basis von:
  - Identität des Akteurs (Benutzers, Prozesses)
  - Identität Objekts (Ressource, auf die zugegriffen werden soll)
  - Zusätzlichen Regeln und Eigenschaften



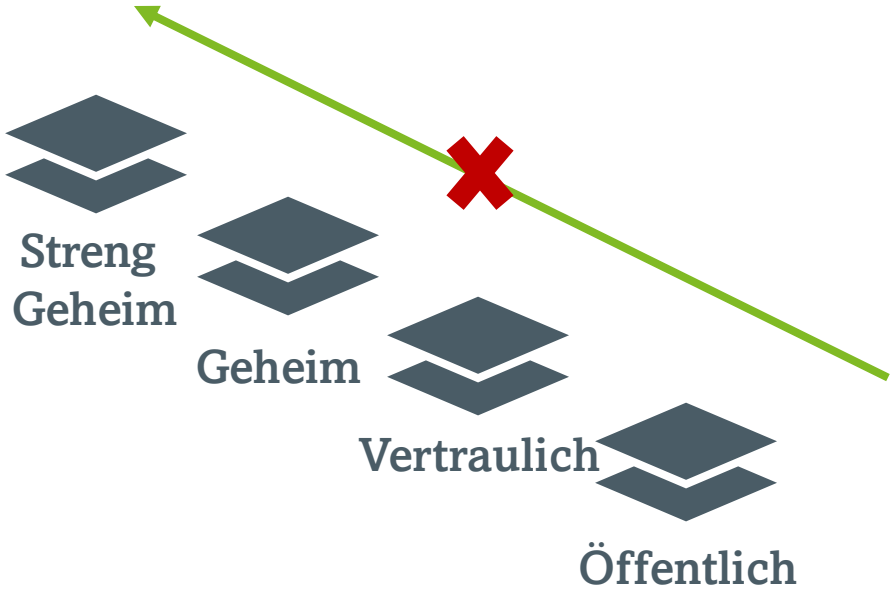
Mandatory Access Control (MAC)

Bekannteste Umsetzung von Mac: Bell-LaPadula-Modell (BLP)

- Einfache Sicherheitsmechanismen zur Verifikation
- Formales Sicherheitsmodell
- Beschränkung des Informationsflusses, d.h. Sicherheitsziel ist die Vertraulichkeit
- Beispiel: Realisierung eines Multilevel Security System (MLS)

Multilevel Security System (MLS)

- Alle Subjekte und Objekte erhalten eine Sicherheitsmarkierung gemäß der Stufe ihrer Vertraulichkeit
- Idee: Höher klassifizierte Daten dürfen nur von Mitarbeitern mit einer ausreichenden Sicherheitsstufe gelesen werden



Multilevel Security System (MLS)

Formalisierung Multilevel Security Systems:

- M sei die Zugriffsmatrix
- SC sei eine Menge von Sicherheitsmarkierungen, und auf SC gelte eine partielle Ordnung
- O: Menge der Objekte, S: Menge der Subjekte

Klassifikation

$o \in O$: $SC(o) \in SC$ sei die Sicherheitsstufe von Objekt o

Spielraum (clearance)

$s \in S$: $SC(s) \in SC$ sei die Sicherheitsstufe von Subjekt s

Multilevel Security System (MLS)

No Read-Up Security-Property:

- $\forall s \in S, \forall o \in O$: **read** $\in M(s, o) \Rightarrow SC(o) \leq SC(s)$
- **Read muss Teil des Feldes M(s,o) sein und es muss gelten, dass die Sicherheitsmarkierungen von o schwächer sind als die von s**
  - Darf s das Objekt o lesen, dann muss Sicherheitsstufe von o kleiner der Sicherheitsstufe von s sein.

No Write-Down Security-Property:

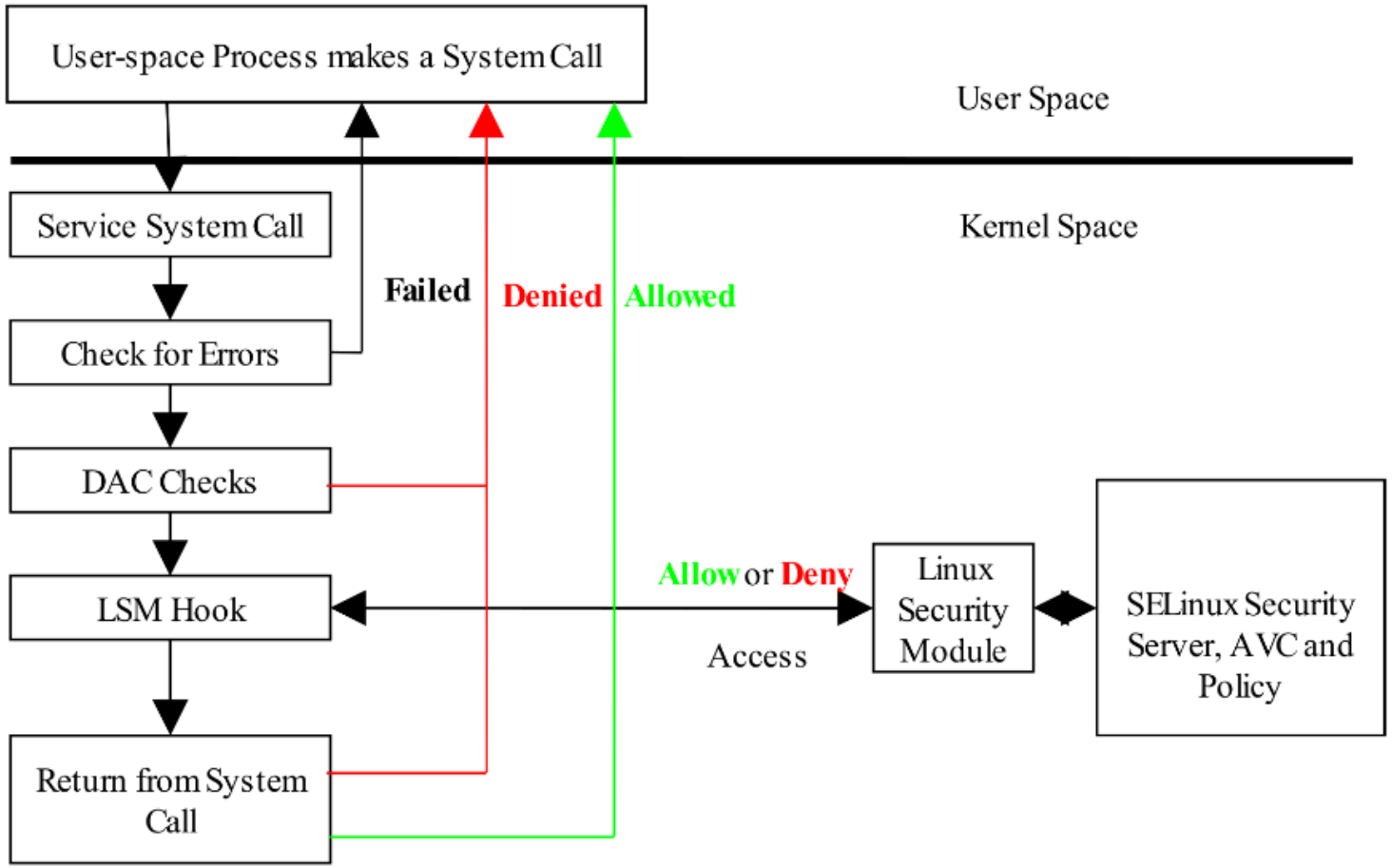
- $\forall s \in S, \forall o \in O$: **write** $\in M(s, o) \Rightarrow SC(s) \leq SC(o)$
- **Write muss Teil des Feldes M(s,o) sein und es muss gelten, dass die Sicherheitsmarkierungen von s schwächer sind als die von o**
  - s darf Zugriff auf das Objekt o nicht unter seine Sicherheitsstufe setzen

Mandatory Access Control (MAC)

Vorteile:

- Viele Betriebssysteme bieten BLP-Erweiterungen

Realisierung der MAC unter SELinux



Mandatory Access Control (MAC)

Vorteile:

- Viele Betriebssysteme bieten BLP-Erweiterungen
- Einfach zu implementieren (Anpassung des Referenzmonitors)
- Formales Modell (man kann Sätze über Eigenschaften beweisen)
- Gut geeignet zum Nachbilden hierarchischer Informationsflüsse:
  - z.B. beim Geheimdienst, Militär

Multilevel Security System (MLS)

Nachteile:

- Beschränkte Ausdrucksfähigkeit: keine Integrität (blindes Schreiben von z.B. neuen Dateien)
- Keine Modellierung von verdeckten Kanälen (covert channels), über die Informationen dann doch unberechtigt fließen können

Verdeckte Kanäle (covert channels)

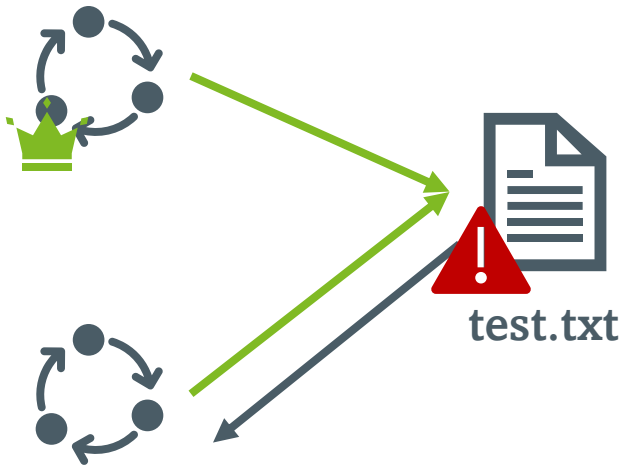
Informationsfluss über einen Kanal, der nicht explizit zur Informationsübertragung vorgesehen ist

Beispiele:

Verdeckte Kanäle (covert channels)

Beispiele:

- Ressourcen-Konflikt: High-Level-Prozess erzeugt eine Datei, so dass ein Low-Level-Prozess eine Fehlermeldung erhält, wenn eine Datei mit gleichem Namen erzeugt wird (1 Bit an Information)



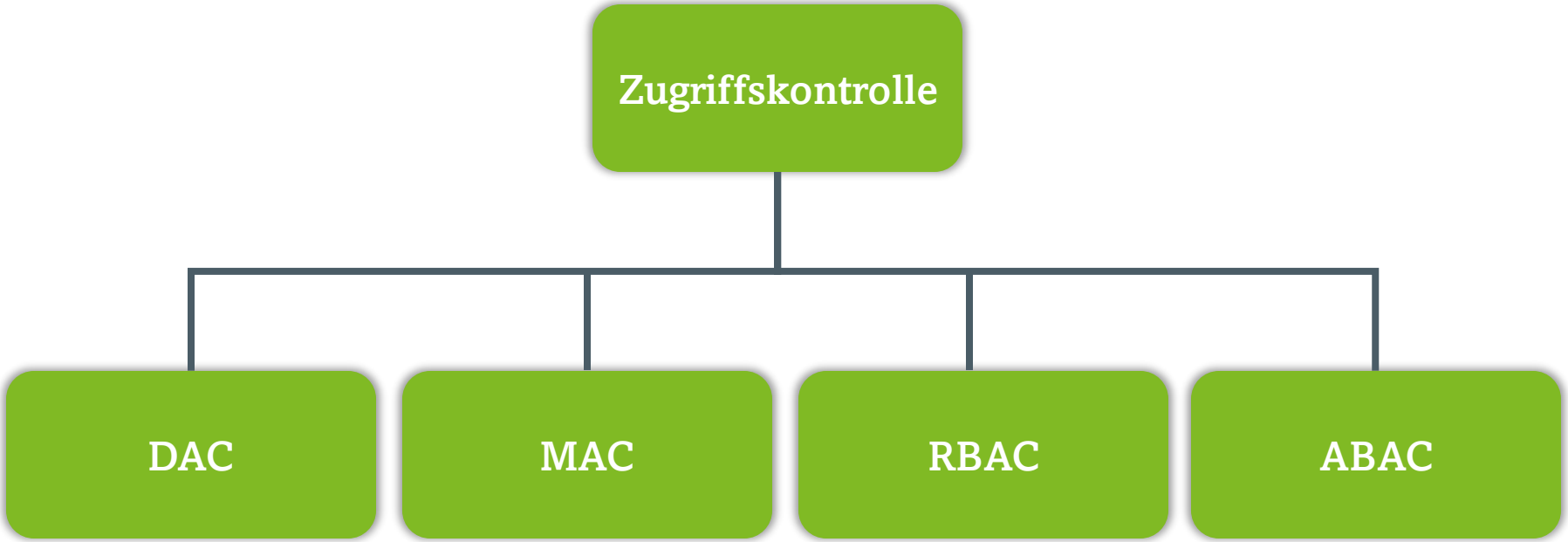
ℹ Datei `test.txt` vorhanden

Verdeckte Kanäle (covert channels)

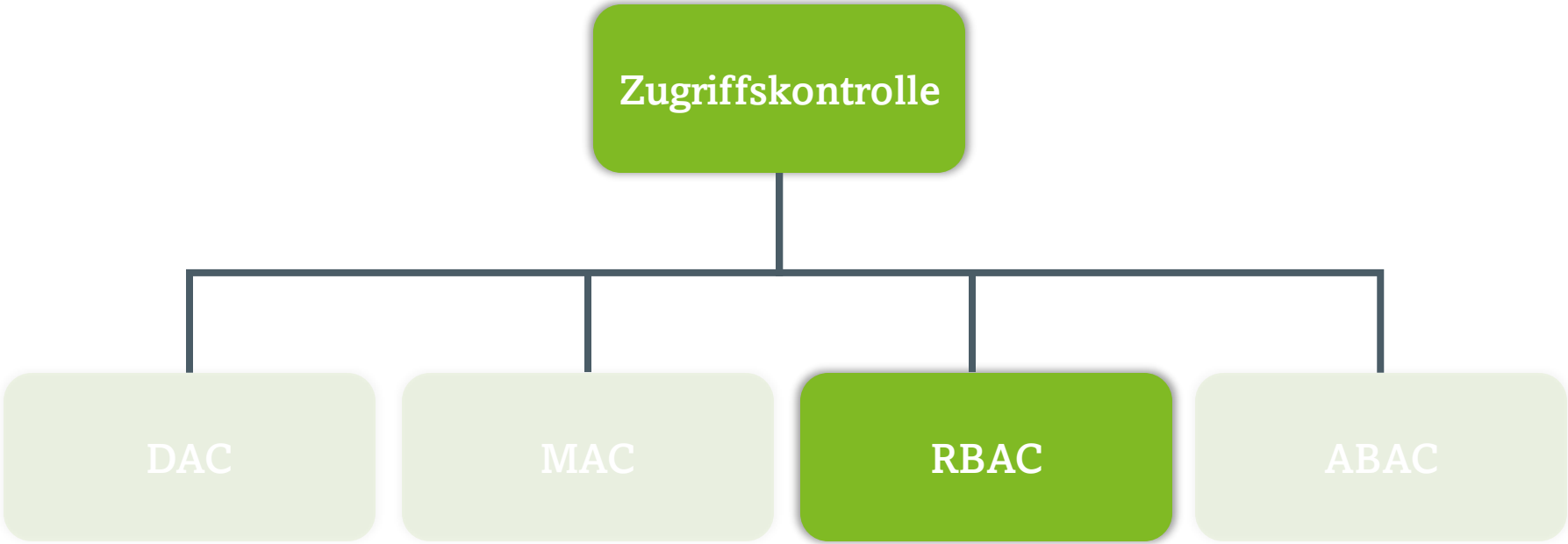
Beispiele:

- Ressourcen-Konflikt: High-Level-Prozess erzeugt eine Datei, so dass ein Low-Level-Prozess eine Fehlermeldung erhält, wenn eine Datei mit gleichem Namen erzeugt wird (1 Bit an Information)
- Gezielte Manipulation von Metadaten, die Informationen kodieren können

Arten der Zugriffskontrollen



Arten der Zugriffskontrollen

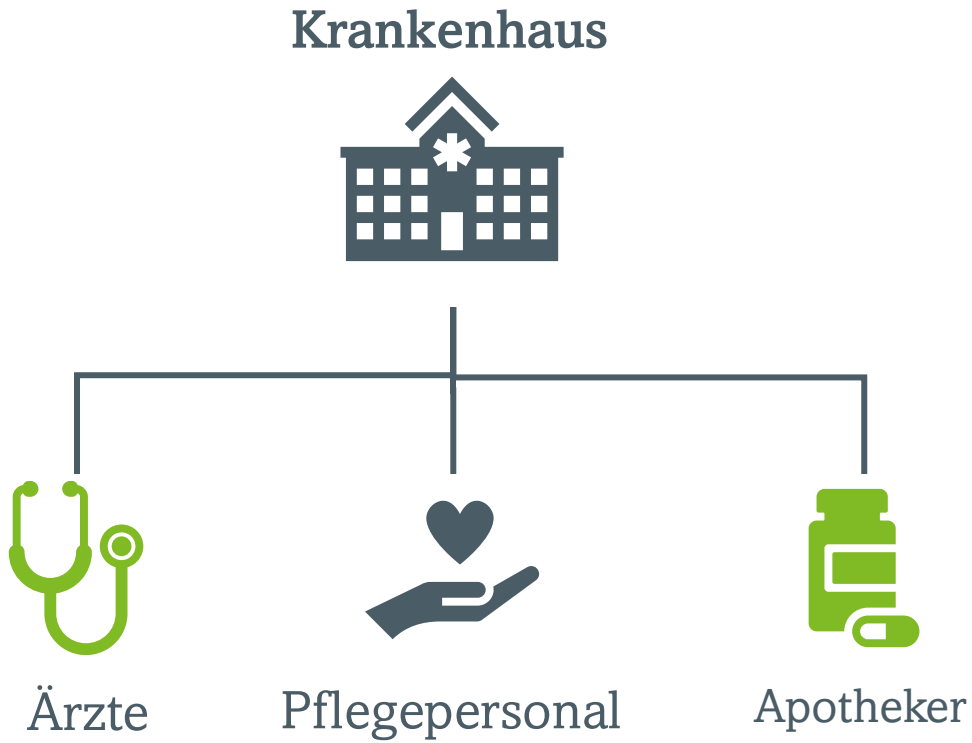


Role Based Access Control (RBAC)

- Rechte werden nicht an Subjekte, sondern an Rollen geknüpft.
- Rollen entsprechen häufig Funktionen/Aufgaben in einer Organisation (Krankenhaus, Bank, Behörden, Unternehmen)
- Gut geeignet für hierarchisch aufgebaute Organisationen

Role Based Access Control (RBAC)

Beispiel:



Role Based Access Control (RBAC)

Vorteile:

- Rollenkonzepte sind flexibel verwendbar
- Vereinfachtes Berechtigungsmanagement
- Abbildung von hierarchischen Organisationsstrukturen
- Intuitive Abbildung auf Geschäftsprozesse
- Möglichkeit, organisationsinterne Sicherheits- und Kontrollregeln abzubilden
  - (z.B. das Konzept der Aufgabentrennung im Bankenbereich)
- Sowohl Vertraulichkeit als auch Datenintegrität
- BLP kann durch RBAC simuliert werden

Role Based Access Control (RBAC)

Nachteile:

- Woher bekommt man überhaupt die Rollen, die für eine Organisation relevant sind? (Problem des Role Engineerings)
- Automatische Zuweisung von Berechtigungen zu Rollen wird nicht ausreichend unterstützt (manuelle Zuweisung in großen Organisationen nicht praktikabel).
- Rollenproliferation – Zu viele Rollen

Arten der Zugriffskontrollen