Sicherheit

Problem der Sicherheit

- Facetten der Sicherheit
  - engl.: *safety* oder *security*
- Safety
  - eher im Sinne „Verlässlichkeit“
  - Wird die Anwendung sicher, d.h. ohne Fehlverhalten ausgeführt?
  - Mögliches Fehlverhalten
    - keine Ausführung
    - Programmabbruch
    - semantische Fehler (falsche Ergebnisse)

Facetten der Sicherheit

- Security
  - wird meist als „Sicherheit“ verstanden
  - betrifft den Schutz vor fremden Einwirkungen, die dazu geeignet sind,
    - die Anwendungsausführung zu beeinträchtigen
    - die Beschaffenheit der Anwendung zu verändern
    - Zugriff auf die in der Anwendung verwalteten Daten zu erlangen (zum Lesen und / oder Verändern)
    - die Zuordnung von Aktionen auf verantwortliche Identitäten zu ändern (Nutzung fremder Identitäten zur Durchführung von Angriffen)

Begriffe

- Angriff
  - eine solche, beabsichtigte Aktion
  - Akteur ist der „Angreifer“
- Identität
  - eindeutige Repräsentation eines Benutzers oder einer Anwendung
  - Personalausweis bei Personen, Registriernummer o.ä. bei Anwendungen
  - in der IT-Welt eine Sammlung von Daten, die eindeutig einem Anwender zugeordnet ist

Der ursprünglich repräsentierte Benutzer muss Sorge tragen, dass kein anderer Benutzer sich durch die Verwendung seiner Identität als er selbst vor dem System ausweist.

Begriffe, Prinzipien

- Identifikation
  - Erbringung des Beweises, dass die von der Identität behauptete Zuordnung zu einer realen Person (Subjekt) korrekt ist
  - Nutzung biometrischer Merkmale
- Authentifikation (auch *Authentisierung*)
  - Erbringung des Beweises, dass die Identität von einem Subjekt verwendet wird, welche das zur Nutzung erforderliche Geheimnis kennt (falls vorhanden)
  - Kennwort, evtl. verknüpft mit Zertifikaten usw.

Problem der Biometrie
Beweis kann (noch) nicht erbracht werden, dass das Merkmal mit dem Subjekt in Verbindung steht.

Problem der Kennwörter
Man kann nur nachweisen, dass das Subjekt das Kennwort kannte.

Begriffe, Prinzipien

- Berechtigungen
  - Besitz der Berechtigung ist eine Bedingung zur Nutzung oder zum Zugriff auf ein geschütztes Gut
  - teilweise auch synonym mit *Privileg* (aber auch zahlreiche Differenzierungen)
- Autorisation
  - Vorgang der Zuteilung von Berechtigungen an einen Benutzer (bzw. an die von ihm eingesetzten Anwendungen)
  - Authentifikation muss vorangehen, wenn Berechtigungen an Identitäten gekoppelt sind
- Zugriffskontrolle
  - Auswertung der vorhandenen Berechtigungen bei einem aktuellen Zugriffsversuch mit Genehmigung oder Ablehnung

Lokales Beispiel

- Beispiel: UNIX
- Identität: Jedem Benutzer ist eine UID (User ID) zugeordnet, textuell als Benutzername repräsentiert
- Authentifikation: Der Benutzer muss das korrekte Kennwort zum gegebenen Benutzernamen liefern
- Autorisation: Der Benutzer darf beliebige Programme starten und auf Elemente des Dateisystems zugreifen, denen er als Benutzer zugeordnet ist
  - außerdem zusätzliche Regelungen für Gruppen und andere
- Zugriffskontrolle beim Starten oder bei Zugriff

Lokales Beispiel

- Delegation von Berechtigungen vom Benutzer auf seine Prozesse
  - Benutzer agiert nicht „selbst“ im System, sondern nur indirekt
  - Zuordnung einer „effective user id“ zum Prozess
  - kann durch „set user-id“ geändert werden (Ausführung unter anderer Identität)

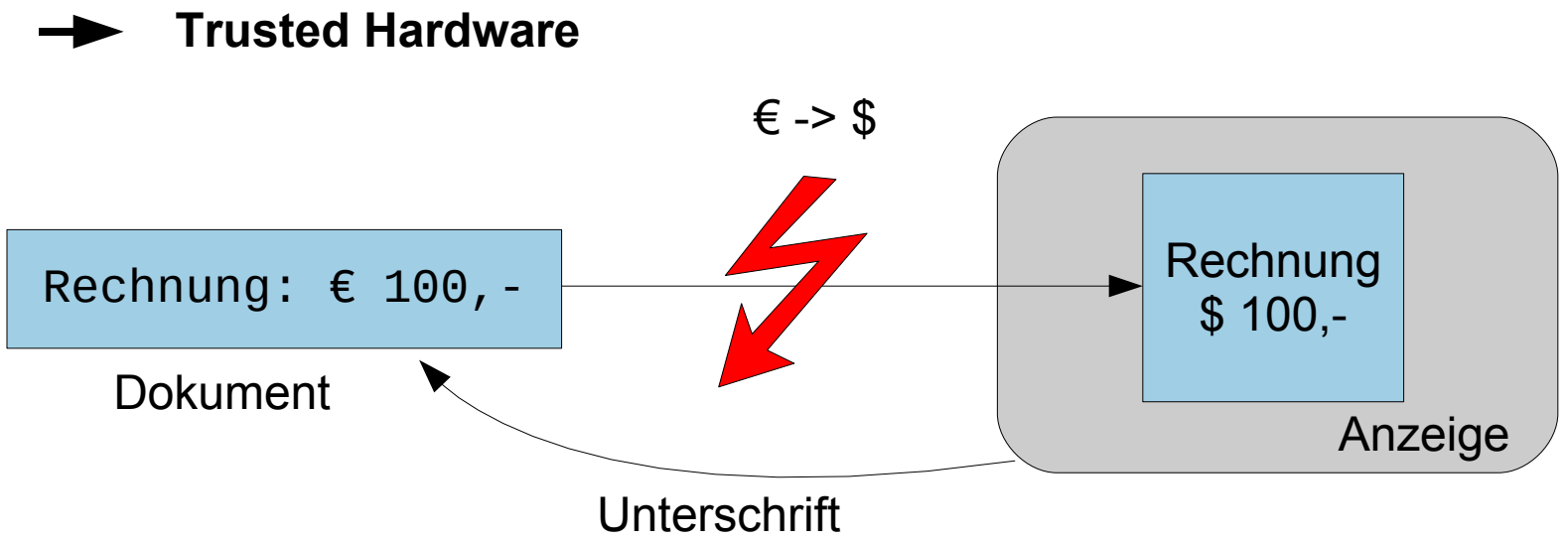
Bedingung an das Betriebssystem
Das BS muss sicherstellen, dass die Delegation von Rechten vom Benutzer an seine Prozesse korrekt ist und dass diese für deren Laufzeit verbindlich bleibt. Der Benutzer oder seine Prozesse dürfen keinen Einfluss auf fremde Prozesse ausüben.

Verteiltes Beispiel

- Lokales Vorgehen überschaubar
  - erfordert keine Kommunikation über evtl. unsichere Kanäle
- Entfernte Dienste, z.B. Webdienste
  - Benutzer muss sich entfernt authentifizieren
- Nutzung von verschlüsselten Kanälen
  - auch schon zur Anmeldung
  - Schlüsselaustausch, TLS
- Autorisation erlaubt Zugriff auf entfernte Ressourcen
  - Server führt Zugriffskontrolle durch

Mögliche Gefahren

- Angriffsmöglichkeiten
  - minimiert durch kryptografische Methoden
  - jedoch muss man eine „vertrauenswürdige Zone“ zur Verfügung haben
  - Privater PC erfüllt diese Bedingung nicht unbedingt!

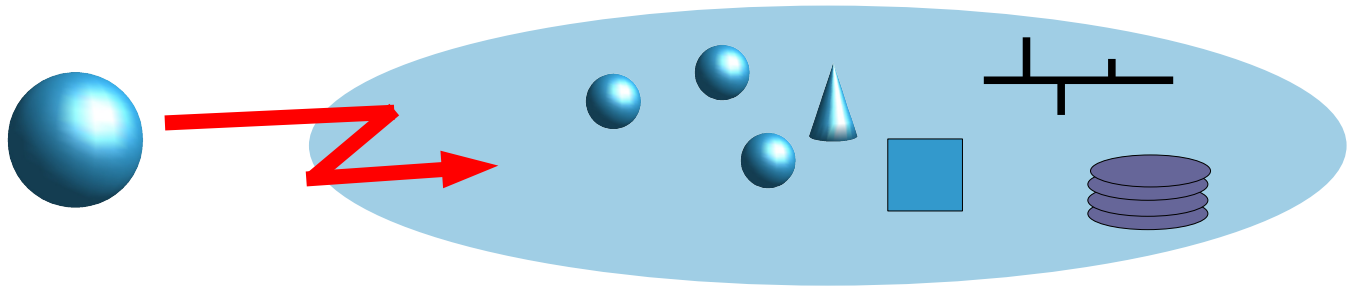


Agentenspezifische Gefahren

- Agentensystemen droht Ungemach
  - Generelle Gefahren: wie für jedes andere IT-System
- Es gibt aber auch spezifische Gefahren
  - Malicious Agent
  - Malicious Host

Der böse Agent

- *Malicious-Agent-Problem (MAP)*
  - Agent, welcher so beschaffen ist, dass er im Namen seines Anwenders Angriffe auf die Umgebung ausführt
- Umgebung besteht aus
  - System (Betriebssystem, Ressourcen, Hardware)
  - Stellen
  - andere Agenten
  - andere Komponenten, die für das Agentensystem relevant sind
  - andere Software, die auf dem System läuft



Bedrohungsszenarien

- Angriff gegen andere Agenten
  - SP: Ausspähung der Daten anderen Agenten
  - SA: Modifikation der Daten der anderen Agenten
  - DoS: Ressourcenaufbrauch, sodass andere Agenten keinen oder nur geringen Fortschritt machen
  - ID: Täuschung anderer Agenten über die wahre Identität
  - MA: Mangelnde Aufrichtigkeit in der Kooperation
- Angriff gegen das System
  - AU: Modifikation der Autorisation
  - AC: Aussetzen / Modifikation der Zugriffskontrolle
  - ID: Täuschung über die wahre Identität
  - DoS: Unbrauchbarmachung des Systems

Maßnahmen gegen Zugriff auf Agenten

- Nutzung von Sprachkonstrukten zur Zugriffssteuerung
  - private-Kennzeichnung (in Java)
- Verwendung einer restriktiven Zugriffspolitik auf Komponenten
  - Ausführung eines Agenten in einem abgeschirmten Bereich mit eingeschränktem Zugriff
  - nur bestimmte Komponenten dürfen auf Agent zugreifen
- Schutz nur gegeben, wenn Stelle selbst gegen den Agent abgeschirmt ist
  - nur geringe Angriffsflächen bieten
  - Stelle besonders sensibel – wenn ein Agent Zugriff erlangt, könnte direkter Zugriff auf alle anderen Agenten erfolgen.

Hilfreich gegen Bedrohungen SP, SA

Maßnahmen gegen Identitätstäuschung

- Kopplung einer Identitätsreferenz an den Agenten
  - welcher dieser selbst nicht ändern kann
  - welche den Agenten eindeutig kennzeichnet
- Interaktion zwischen Agenten
  - Empfänger einer Nachricht möchte wissen, ob der Absender bestimmte Rechte hatte
  - Leicht bei Kommunikation auf Applikationsebene: Markierung von Interaktionen mittels dieser Identitätsreferenz
  - Schwierig bei Methodenaufrufen: Jede Methode müsste als einer der Parameter die Identitätsreferenz erwarten
- Kopplung dieser Identitäten erlaubt Nachvollziehbarkeit, welcher Anwender eine Aktion veranlasst hat

Maßnahmen gegen Identitätstäuschung

- Unterscheidung zwischen Identität des Anwenders und des Agenten
- Identität des Anwenders: wichtig für Zurechnung von Aktionen
  - Agenten selbst sind (bislang) nie die eigentlichen Urheber der Angriffe
  - Interesse der jeweiligen Anwender relevant
  - Anwender müssen unterscheidbar gekennzeichnet sein
- Identität des Agenten: wichtig als technische Referenz
  - Agentenidentität sollte eindeutig sein
  - alternativ: Der Identität muss zumindest eindeutig die Identität des eigenen Anwenders zuzuordnen sein
    - als Beispiel: ID des Anwenders=12345, ID des Agenten=12345.6
    - Änderung im vorderen Teil ist unzulässig

Maßnahmen gegen Identitätstäuschung

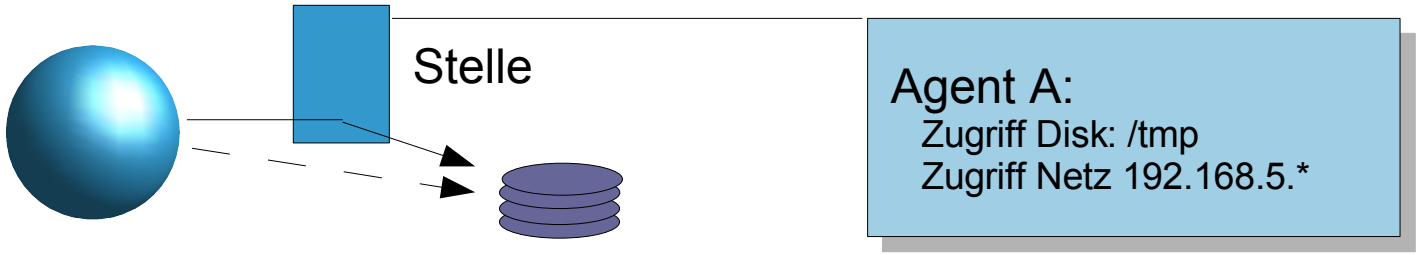
- Damit „darf“ eine missbräuchliche Verwendung der Anwendung desselben Anwenders schaden, nur keiner fremden.
- Hintergrund der „Wahlfreiheit“
  - Agenten kommen auf das eigene, lokale System
  - Agenten tragen eine Kennung (Agentenidentität) mit sich
  - Die Authentizität der Kennung ist vom eigenen System aus nicht nachweisbar! (könnte konstruiert sein)

- andere Möglichkeit: signierte Kennungen
  - Kennungen werden von einer vertrauenswürdigen Instanz ausgegeben
  - Agenten können Kennung nicht selbst modifizieren
  - allerdings erhöhter Aufwand, wenn viele Agenten zum Einsatz kommen

Hilfe gegen ID-Angriff

Szenarien des Autorisationsangriff

- Angriff auf die Autorisation und Zugriffskontrolle
  - Agent ist also bereits authentifiziert und möchte weiter gehende Rechte, als ihm zustehen
- Angriffsmöglichkeiten für den Agenten
  - direkt
    - Bei exponierten Ressourcen (solchen, die der Agent direkt ansprechen kann): Angriff auf Ressourcenzugriffssteuerung des Betriebssystems
  - indirekt
    - Rechtevergabe der Stelle beeinträchtigen



Zugriffskontrollliste

Maßnahmen gegen Autorisationsangriff

- Gegenmaßnahme
  - Verhinderung eines Zugriffs auf die sensiblen Komponenten durch den Agenten
  - bei Objekt-orientiertem Entwurf: Kapselung des Autorisierers, Referenz privat
  - Zugriff auf Ressourcen muss von der Laufzeitumgebung geregelt werden (z.B. in Java: *SecurityManager*, „Sandbox“)

Hilfe gegen AU/AC-Angriff

Szenarien und Maßnahmen bei DoS

- Agent versucht die Ausführung anderer Agenten zu be- oder verhindern (Denial-of-Service)
- Aufbrauchen wichtiger (begrenzter/unteilbarer) Ressourcen
  - CPU
  - Speicher
  - Synchronisation (Monitore)
- Ausnutzen von Stabilitätsschwachpunkten
- Abhilfe
  - Ressourcenkontrolle
  - Stoppen eines Agenten, der Ressourcen im Übermaß nutzt
  - Frühzeitiges Erkennen von „gefährlichem“ Code

Hilfe gegen DoS

Denial-Of-Service-Angriff

- Schwer zu verhindern
- Vom Schweregrad in vielen Fällen geringer als bei den anderen Übergriffen
  - außer bei Anwendungen, deren Verfügbarkeit garantiert sein muss
  - es werden keine Modifikationen oder Ausspähungen begangen
  - einfach lästig
- Gegenmaßnahme
  - Aussperren aller Agenten eines bestimmten Typs, einer bestimmten Quelle, eines bestimmten Autors

ABER...

- All diese Maßnahmen hängen von der verwendeten Laufzeitumgebung ab
  - Schutzmechanismen des Systems zur Compile-Zeit können zur Laufzeit unwirksam sein (z.B. Zugriff auf private Variablen)
  - Reaktion des Systems bei Laufzeitfehlern
    - Wird nur der defekte Agent gestoppt?
    - Stürzt das ganze System ab?
    - Abhängig von Ausnahme-Verarbeitung (Exception handling)
- Keine allgemein gültigen Schutzmaßnahmen
- Jedes System trägt seine eigenen Verwundbarkeiten
  - insbesondere auch Java

Java-spezifische Probleme

- Zahlreiche spezifische Schwachstellen von Java
- Malicious-Agent-Problem ist nicht alleinige Ursache der Probleme
  - leider nicht nur Bugs, sondern auch Designfehler in Java
  - Vielleicht mögliche Korrektur in späteren Java-Versionen?

Java ist **keine** optimale Wahl, was die Erstellung eines sicheren Agentensystems angeht.

Java erlaubt **nicht** das System **immun** gegen bösartige Agenten zu gestalten.

Das Finalizer-Problem

- Finalizer
  - Thread, welcher für die Entsorgung nicht mehr referenzierter Objekte sorgt
  - Verhalten eines Objekts ist durch die Methode **finalize()** steuerbar
    - finalize() wird aufgerufen, wenn der Entsorger (Garbage collector, GC) das Objekt entfernen will

> ... The finalize method may take any action, including making this object available again to other threads; the usual purpose of finalize, however, is to perform cleanup actions before the object is irrevocably discarded

JDK-Dokumentation 1.5

- Notwendig, um das Java-Laufzeitsystem sicher (stabil) zu machen
  - z.B. vergessenes close() bei Strömen wird über finalize() aufgerufen

Das Finalizer-Problem

- Schwachpunkt, da die Implementierung durch den Angreifer möglich ist

```
public void finalize() {
    while (true) ;
}
```

Hängt den Finalizer auf
und führt zu hoher Systembelastung
(DoS)

- Zugriffskontrolle
  - Finalizer gehört zu den Systemkomponenten
  - Agentensystem könnte diesem Thread erweiterte Rechte zubilligen

Synchronisationsproblem

- Benutzung von Monitoren ist ein Schwachpunkt
  - Möglichkeit, auf beliebigen Objekten und Klassen zu synchronisieren – auch auf Systemklassen

```
synchronized(Thread.class) {
    while (true) ;
}
```

Hängt das Laufzeitsystem auf, wenn es einen neuen Thread erzeugen will

Threadterminierungsproblem

- Threadterminierung in Java ist asynchron und basiert auf dem Werfen eines „Throwables“
  - Throwables kann man fangen!

```
while (true) {
   try {
       while (true);
   }
   catch (ThreadDeath td) {}
}
```

- Man kann Threads in Java nicht verlässlich stoppen!
- **Thread.stop** soll nicht mehr verwendet werden (schafft inkonsistente Objektzustände)

Threadterminierungsproblem

- Empfehlung von Javasoft

> Most uses of **stop** should be replaced by code that simply modifies some variable to indicate that the target thread should stop running. The target thread should check this variable regularly, and return from its run method in an orderly fashion if the variable indicates that it is to stop running.

JDK-Dokumentation 1.5

- Dieses Konzept ist für bösartigen Code nicht anwendbar
- Vogel-Strauß-Politik
  - Man hat ein Interesse, dass das Applet beim Kunden funktioniert, nicht, dass es den Browser angreift (dann kommt der Kunde nicht wieder)

Weitere Java-Schwächen

- Schwächen auf Bytecode-Basis
  - Modifizierter Bytecode
  - Bytecode-Verifizierer kann diesen Code trotz Modifikation genehmigen
  - Beispiele
    - Verhinderung der Threadbeendigung
    - Verhinderung der Rückgabe des Monitors
- Zwischenspeicher (Heap) mit schwacher Absicherung
  - Überlauf bringt ganze Java-VM zum Abbruch

Semantischer Angriff

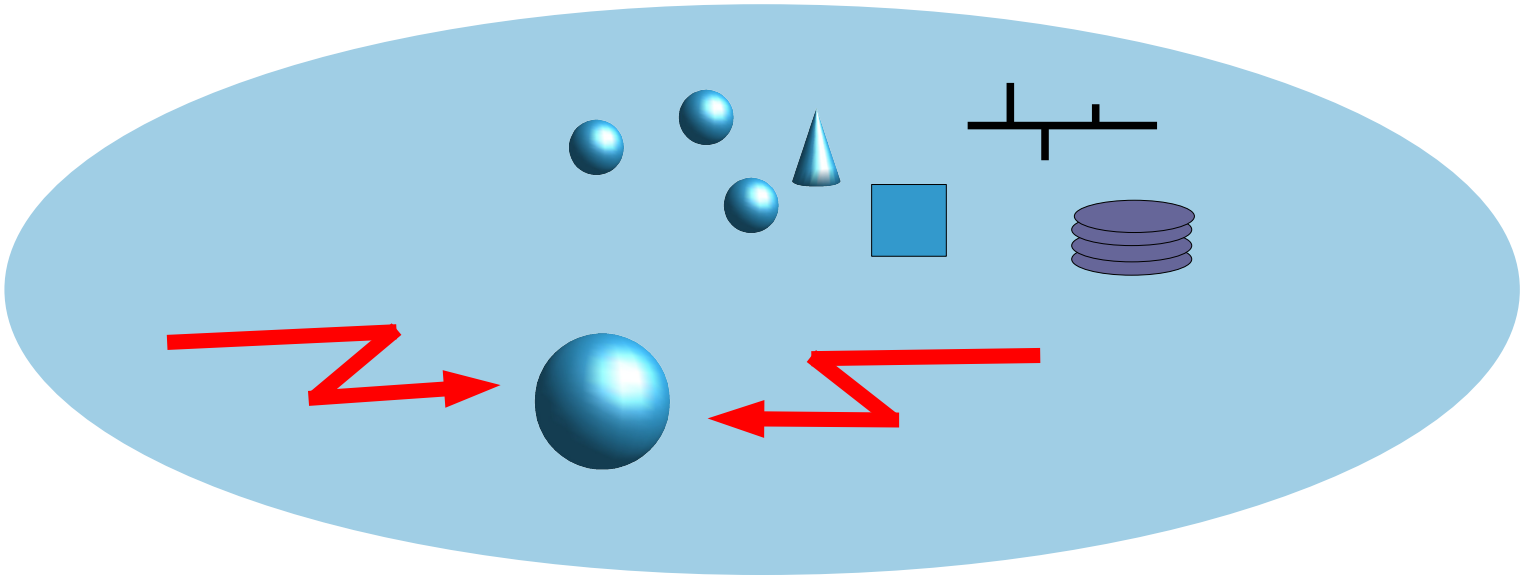
- Fehlende Wahrhaftigkeit
  - Agent „betrügt“ andere Agenten oder die Stelle
  - Beispiel elektronischer Handel
    - Falsche Angaben zum Anwender
    - Falsche Angaben zum bisherigen Ablauf
  - Beispiel kooperative Agenten
    - Agent leugnet Wissen (z.B. über andere Agenten, die eine bestimmte Funktion leisten könnten)

Semantische Angriffe lassen sich prinzipiell nur in seltenen Fällen automatisch entdecken und behandeln.

- Nicht versuchen, Grundsatzprobleme zu lösen, die gar nicht agentenspezifisch sind!

Die böse Stelle

- Neuartige Bedrohung
  - Nicht der Agent ist der Angreifer, sondern die Umgebung!

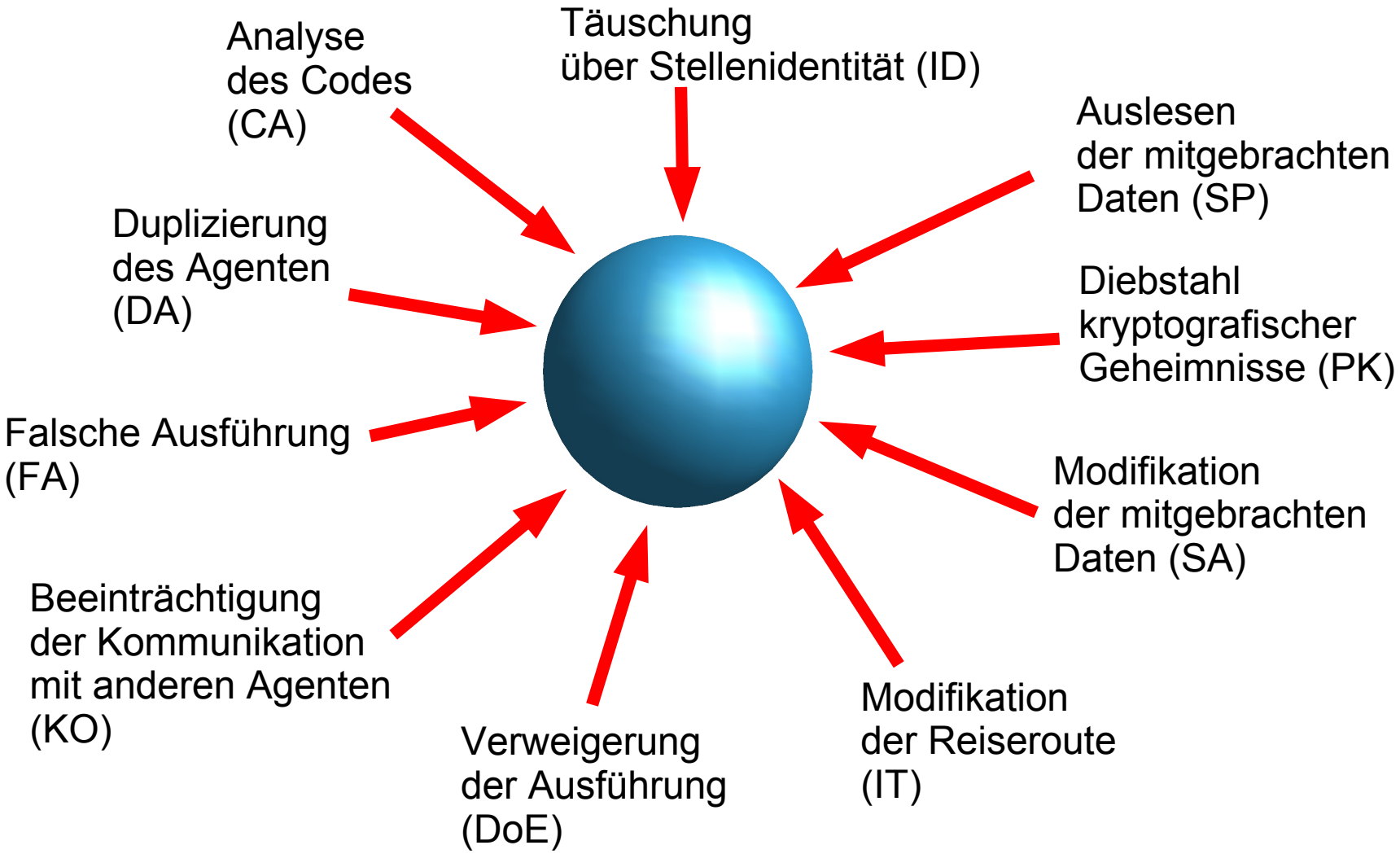


Malicious-Host-Problem (MHP)

Bösartige Stellen

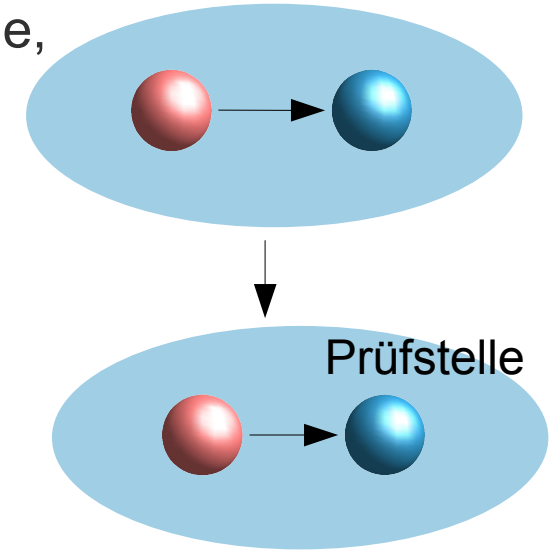
- Szenario
  - Agent migriert zum Erwerb eines Guts auf eine Stelle des elektronischen Marktplatzes; führt „Geld“ mit
  - Agent kehrt zurück
    - Geld ausgegeben
    - Ware nicht bestellt
    - Kreditkartendaten weitergegeben
    - Zeitschriftenabonnement abgeschlossen
  - Aber laut Code des Agenten hätte all das nicht passieren dürfen!
    - Agent läuft in eigener Umgebung fehlerfrei
    - Offenbar wurde der Agent Ziel eines Angriffs

Bedrohungsszenarien



Wird der Agent korrekt ausgeführt?

- Szenario
  - Agent erreicht eine Stelle, wird ausgeführt
  - Ist die Ausführung korrekt?
- Prüfung
  - Wiederholung der Ausführung an einer zweiten Stelle, die vertrauenswürdig ist
  - Vergleich der Ergebnisse
- Erforderlich
  - Korrekte Erfassung der Start- und Endzustände
  - Korrekte Erfassung der Eingabewerte
  - Korrekte Übertragung an die Prüfstelle
  - Einsatz von Signaturen zum Schutz vor Veränderung



Korrekte Ausführung

- Problem
  - Wie vertrauenswürdig ist diese Stelle? Sind die übermittelten Werte wahrhaftig?
  - Prüfstelle muss auf jeden Fall vertrauenswürdig sein
- Kryptografische Spuren
  - Ausführung des Agenten auf den jeweiligen Stellen muss im Log nachvollziehbar sein
  - Über Logausschnitt wird ein Hash gebildet, signiert und hinterlegt
  - Signierter Loghash wird außerdem dem Agenten mitgegeben
  - Anwender kann im Nachhinein die Ausführung nachspielen und damit überprüfen

Kann man sicher sein, dass der Angriff auf den Agenten im Log abgebildet wird?

Blackbox

- Agent als „black box“
  - Verhalten ist nicht einsehbar
  - Daten sind nicht einsehbar
  - Daten sind nicht modifizierbar

- Aber die fremde Stelle
  - verfügt doch über den Code!
  - kann über Analyse Zugriff auf einige der Felder erlangen
  - kann ggf. aus dem serialisierten Zustand und der Klassenstruktur Rückschlüsse ziehen

Time-Limited Blackbox Security

- Blackbox-Eigenschaft nur für eine „Weile“ aufrecht erhalten
  - so lange, bis der Agent weiter gewandert ist
  - in der Verweilzeit muss es der Stelle unmöglich gemacht werden, den Code zu analysieren

Fritz Hohl, MOLE-System, Stuttgart

→ **Code obfuscation**

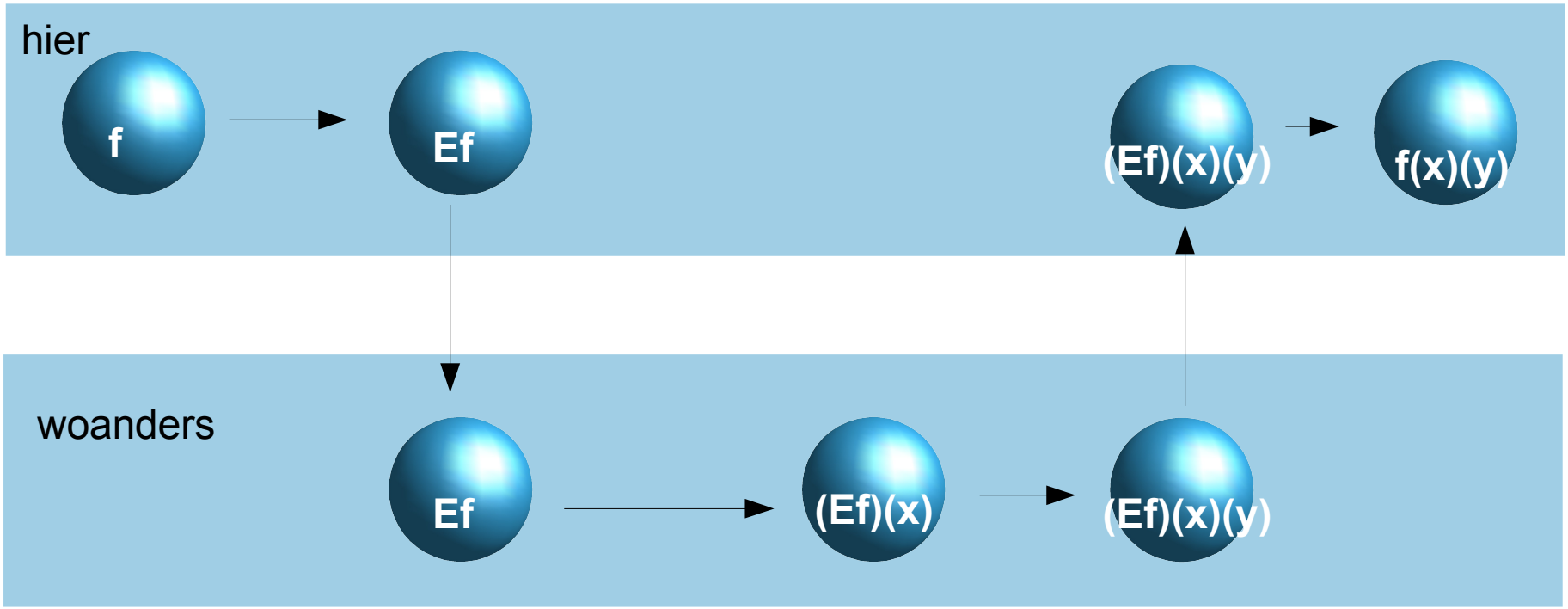
- Code wird so umstrukturiert, dass er
  - seine eigentliche Funktion verbirgt
  - korrekt ausgeführt wird

Dieser Code ist nicht mehr wartbar; die Analyse der Funktion von „geordnetem Code“ ist ohnehin schon schwierig auf algorithmischem Wege durchzuführen.

→ Kein großer Zugewinn

Verschlüsselte Funktionen

- Idee
  - Funktion wird „verschlüsselt“ berechnet, keine Entschlüsselung während der entfernten Abarbeitung erforderlich



Verschlüsselte Funktionen

- Verfahren taugt nur für Klasse der Polynome
  - Verschlüsselung verhält sich treu: (Ef)(x) =E(f(x))
- Möglichkeit, Verschlüsselung ohne Entschlüsselung zu betreiben
  - Versendung kryptografischer Geheimnisse (privater Schlüssel, Kennwort) wird vermieden
- Für allgemeinen Fall nicht anwendbar

Gesicherte Umgebungen

- „Tamper-resistent Hardware“
  - spezielle Umgebung, die zertifiziert ist
  - zugesichert, dass der Agent dort korrekt ausgeführt wird
  - Hardware + BS + Umgebung festgelegt
- Tiefere Schichten können den Agent erst entschlüsseln
  - Höhere Schichten müssen den Agenten nach unten unverändert durchgeben
- Nachteile
  - Unflexibles System
  - Wie spielt man Korrekturen ein?
  - Preis
  - Wird der Agent wirklich auf dieser Umgebung ausgeführt?

Bewertung der Gegenmaßnahmen

- Wirksamkeit gegen einige der genannten Angriffe
  - korrekte Ausführung kann so zugesichert werden, falls der Zertifizierer vertrauenswürdig ist
- Angriffe semantischer Natur
  - der Agent könnte ja „später“ an die Reihe kommen (aber wann?)
- Maßnahmen lassen Multiagentensystem-Situation völlig außer Acht
  - Kommunikation mit anderen Agenten / Diensten
  - Deren Falschverhalten ist für den Agenten ein MHP
  - Beispiel: Agent bestellt Buch bei einem Händleragenten
    - aber bekommt etwas ganz anderes

Ist das Malicious-Host-Problem lösbar?

- Vermutung: **nein!**
- MHP ist erneut ein Versuch, grundlegende Probleme mit Agenten lösen zu wollen, weil man meint, sie seien neu
  - Vertrauen in fremde Umgebungen
  - Wahrhaftigkeit der anderen Komponenten, keine semantischen Angriffe
- Falsches Bild eines Agenten. Fakt ist:

Ein Agent ist keine autonome Einheit aus Sicht des Gesamtsystems.
Er kann sich nicht selbst ausführen.
Er wandert nicht selbst.
Er wird zum Teil eines fremden Systems, das ihn aufnimmt.
Er ist ein Strom von Daten, der aus einem Socket herauskommt.

Das MHP

- Problem: Anthropozentrische Vorstellung von Agenten als Abgesandte, die eine Art elektronisches, unabhängiges Individuum darstellen
- Ein migrierter Agent verändert das Verhalten des Wirtssystems, indem er **Teil dieses Systems** wird
- Anhand der Netzverbindung lässt sich praktisch nicht herausfinden, was auf der anderen Seite auf den Agenten wartet
  - echte Stelle?
  - Simulation?
- Lösung über das Ruf-Prinzip (Reputation)
  - Bewertungssystem für Stellen

Sicherheitsprobleme - Fazit

- Sicherheitsprobleme verteilter Systeme verschärft bei Agenten
  - neue Dimension der Bedrohung: MHP
- Defizite bei existierenden Plattformen
  - großer Aufwand notwendig, um einige der Defizite von Java auszubügeln
  - jedoch keine Ressourcenkontrolle, also MAP nicht vollständig gelöst

Können reale Produktivsysteme sich solche Sicherheitsprobleme leisten?