Anatomie Capability basierter Betriebssysteme

joerg@zilium.de

Begriffe:

- Subjekt
  - Dinge, die aktiv irgendetwas tun wollen
  - z.B. ein Prozess
- Objekt
  - eine Ressource
  - wird von einem Subjekt benutzt oder angefordert
  - z.B. Dateien
  - kann bei IPC/RPC auch ein anderer Prozess sein

Subjekte werden mit einer ID versehen

- UNIX: gewoehnlich UID und GID
- Windows: SID -- sie verweist auf weitere IDs

Jedes Objekt hat eine Liste mit Zugriffsrechten

Die Liste hat Eintraege der Form (ID, Zugriffsrechte)

- Windows: bel. Liste mit Zugriffsrechten
- UNIX: 3 Eintraege: Owner, Group, Others
  - Erweiterungen fuer mehr Eintraege
  - stehen teilweise zur verfuegung (Posix 1003.1e)

Problem:

- alle Programme die ich starte, koennen
- alle meine Dateien lesen

Loesungsversuch:

- neue ID fuer jedes Arbeitsaufgabe

Aber: Mehr IDs =>

- mehr Benutzungsaufwand
- mehr Administrationsaufwand
- laengere ACLs an den Objekten
- => Laufzeitverhalten

Ergo:

- Je mehr Sicherheit Er will, umso mehr Aufwand er hat
- Kein "least privilege"

Confinement:

- Kann ein Subjekt unberechtigt Daten kopieren/verschicken?
- Zugriffskontrolle andersherum

Anwendungsbeispiele:

- selbst ein trojanisches SSH koennte seinem Herrchen keine Mitgesnifften Daten liefern.
- Alle Programme, die mit vertraulichen Daten hantieren.

Confinement mit AC

ACLs koennen niemals wirklich "Confinement" liefern:

es kann jederzeit ein neues Objekt erstellt werden, auf welches das fragliche Subject Schreibrechte hat.

Fazit:

Sowohl

- "principle of least privilege", als auch
- "confinement"

sind wuenschenswert in einer Welt mit _nicht_ ausschliesslich vertrauenswuerdiger Software.

Keines von Beiden kann man mit ACLs erreichen.

Capabilities sind "Aufrufschluessel".

Ein Subjekt kann nur genau die Objekte benutzen, fuer die es eine Capability hat.

Eine Capability ist _notwendig_ und _hinreichend_ um ein Objekt benutzen zu koennen.

Wenn Capabilities die Schluessel fuer alle Objekte sind muss man verhindern, dass sie einfach
von Subjekten erzeugt und ausprobiert werden.

Dazu gibt es 3 Methoden:

- Tagged Capabilities
- Partitioned Capabilities
- Sparse Capabilities

Tagged Capabilities:

Spezielle Hardware stellt sicher, dass nur als Cap. markierte Daten als Capabilities verwendet werden koennen. Auf Daten, die mit Cap. gekennzeichnet sind, koennen dagegen keine arithmetischen Operationen ausgefuehrt werden.

Offenbar muss jede Speicherstelle und jedes Register ein solches "tag" Bit besitzen.

Partitioned Capabilities:

Capabilities und Daten werden getrennt voneinander gespeichert.
Es gibt spezielle Capability Register, die nur aus speziellem Cap. Speicher geladen werden koennen.

Dies laesst sich mit wenigen Seiten Supervior Code im Kernel auf fast allen aktuellen CPUs emulieren

Sparse Capabilities:

Capabilities sind bei dieser Methode recht "lang":
Nur ein verschwindend kleiner Bruchteil aller moeglichen Capabilities sind gueltig.
Ein boeswilliges Subjekt muess sehr viele Cap. wuerfeln, bis eine gueltige herrauskommt.

Moegliche Umsetzungen:

- crypt. signierte Cap.
- password capabilities

the birth of a

ca

Wo kommen Capabilities her?

Ein Subjekt, das ein neues Objekt erzeugt, bekommt
die (einzige) Capability dazu.

Diese kann dann z.B. IPC auch an andere Subjekte
weitergereicht werden.

Aber wo kommen die Capabilities nach einem Neustart her?
Irgendwie muss man sie doch zugreiffbar abspeichern?

Mann kann "Persistence" benutzen. Es draengt sich geradezu auf.
(tolle Idee!)

Persistence:

Das OS und/oder die Hardware sorgen dafuer, dass von dem ges. Hauptspeicher regelmaessig "Snapshots" gemacht werden.

Bei einem Neustart wird einfach der zuletzt gespeicherte Snapshot wieder hergestellt.

Somit:

- Anwendungen bemerken Neustarts nicht
- 2nd storage wird zu einem grossen Swapspace.
- Hauptspeicher wird zum grossen Cache

Cap. und principle of least pri

Offenbar unterstuetzen Capabilities das "principle of least privilege"

Jedes Subjekt bekommt genau die Capabilities, die es benoetig um seine Aufgaben zu erfuellen

Wenn man einem Subjekt nur auf genau die Objekte
Schreibrechte gibt, in die es erlaubt ist zu schreiben, scheint man auch das Confidence Problem geloest zu haben...

...aber: (!!)

Man kann so _nicht_ verhindern, dass das Subject durch ReadOnly Capabilities an weitere Write-Capabilities kommt.

Moegliche Loesung:

Man fuehrt neben den "read" und "write" Capabilities eine "read/weak"-Capability ein.

Read/Weak Capability:

- erlaubt lesend auf Objekte zugreifen
- gelesene Capabilities werden zu "Weak" umgeformt

Fuer durch weak-Capability gelesene Capabilities gilt:

- write -> NIL
- read -> read/weak
- read/weak -> read/weak

Gibt man einem Subjekt also nur "read/weak" Capabilities, so kann man sich sicher sein, dass es _niemals_ irgendwas schreiben/aendern kann.

Es darf immer nur zu Objekten sprechen, fuer die es expliziet eine write-Capability bekommen hat.

Historie:

- 1966 von Dennis und Van Horn beschrieben

Implementationen:

- B5000
- CAL
- HYDRA
- CAP
- IBM System 38
- i432
- Plessey S250

aktuelle Implementationen:

- Amoeba
  - Benutzt Capabilities nur um POSIX zu emulieren
  - Capabilities for distributed computing
- Mungi
  - benutzt password capabilities
  - single address space
  - wo ist der source? sollte GPL werden
- EROS
  - benutzt Partitioned Capabilities
  - verblueffend performant

EROS ihm seine Capability Typen:

Number
- positive ganze Zahl
- [0 , 2**96-2**16)
- Operationen: get_value und typeof
- werden intern benutzt

Data Page
- ReadOnly und ReadWrite Variante
- aus diesen wird der Adressraum aufgespannt
- Page Size: i386=^4k

Capability Page

- ReadOnly,ReadWeak,ReadWrite Varianten
- speichert Capabilities
- auch aus diesen wird der Adressraum gebaut

Node

- wie eine kleine Cap. Page
- kann nur 32 Cap. aufnehmen
- effizienter Speicher
- ReadOnly,ReadWeak,ReadWrite und Opaque

Start Capability

- einen Prozess aufrufen

Bei der Erzeugung einer Start Capability gibt man eine ID zwischen 0 und 2**16 an.
Dem aufgerufene Prozess wird diese Zahl mituebergeben.

Prozesse sind immer in einem von drei Zustaenden:

- running
- waiting
- available

running

der Prozess wird ausgefuehrt bzw.
wartet auf die Zuteilung einer CPU

Prozess

St

waiting

der Prozess hat mit "call" eine Start-Capability (d.h. einen anderen Prezess) aufgerufen und wartet
auf eine Antwort.

Prozess

St

available

der Prozess hat "return" aufgerufen und wartet darauf von einem anderen Prozess aufgerufen zu werden.

Start

Ca

Man kann drei Dinge mit einer Start-Capability tun:

- call
  - aufrufen und auf rueckkehr warten
- send
  - aufrufen und weiterlaufen
- return
  - aufrufen und darauf warten aufgerufen zu werden

Dem aufgerufenen Prozess wird uebergeben:

- 4 Capabilities
- 64k Daten

wurde "call" zum Aufruf benutzt, so ist eine der 4 Capabilities eine automatisch generierte "return" capability.

Alle Prozessdaten werden in Nodes gespeichert, auch Register, Cap.Register etc.

Der Erzeuger eines Prozesses muss selber den Platz dafuer zur Verfuegung stellen.

Der Kernel ist vollkommen "state less".

Links zum weiterlesen:

- http://www.eros-os.org/
- http://www.zilium.de/joerg/capbib.html