Proseminar
Konzepte von Betriebssystem-Komponenten (KVBK)
Schwerpunkt Sicherheit

Vortragender: Marcel Beister (Beister@gmx.de)
Thema: Superuser vs. dedizierte Capabilities (Trusted Solaris)

1. Intro: Das Problem der Zugriffskontrolle

Im Prinzip geht es bei Zugriffskontrollen um folgendes: Welche Objekte sind einem Subjekt zugänglich und in welcher Form. Mit Objekten sind Ressourcen wie etwa Dateien, Geräte, Netzwerke, Pipes, andere Prozesse und viele weitere Dinge gemeint, wogegen Subjekte normalerweise Programme bzw. Benutzer sind. Mit Zugänglichkeit meint man die Art der Operation die auf die Objekte angewendet werden können, wie beispielsweise das Lesen und Schreiben einer Datei oder die Kommunikation mit anderen Programmen. Im Bezug auf das Dateisystem gibt es im wesentlichen 4 Aspekt: Zugriff verhindern, limitieren, gewähren und wieder entziehen.

Allerdings ist der Dateizugriff nicht das einzige Problem, denn es gibt noch viele andere Dinge die beschränkt und kontrolliert werden müssen und die sich nicht einfach auf Dateizugriffe abbilden lassen. Beispiele hierfür...

- Gerätetreiber oder Kernmodule laden
- Direkter Zugriff auf Hardware
- TCP/UDP-Ports < 1024; rohe IP-Pakete senden (z.B. für Ping)
- Netzwerkkonfiguration (Ethernetadresse, IP-Adresse, Routing...)
- Scheduling ändern (z.B. Realtime-Schedular)
- Signale an fremde Prozesse schicken (-> Prozesse abschiessen, anhalten, ...)
- Zugriffe auf Pipes, Shared Memory, ...

Da man als Person in der Praxis über Programme agiert (z.B. über Konsole) ist einen Kontrolle der Programm-Zugriffsrechte ausreichend, einen Unterscheidung zu Personen also nicht nötig. Eine Person kann mit verschiedenen Logins im System sein und das gleiche Programm kann mehrmals zur gleichen Zeit von verschiedenen Benutzern und mit verschiedenen Rechten aktiv sein.

2. Die konvetionelle Lösung: Superuser

Der klassische Ansatz bei Unix im Bezug auf den Dateizugriff sind die normalen Zugriffsrechte für User, Group und Other (siehe SP1), wobei diese Einteilung grundsätzlich ziemlich grob ist. Für die nicht dateibezogene Zugriffskontrolle ist die konventionelle Lösung das „root-sein-oder-nicht-sein“, d.h. im Klartext: Um auf diverse Geräte, Prozesse oder Netwerkports zugreifen zu können benötigt man Root-Rechte, sonst geht nichts. Der Kernel verwaltet all diese Ressourcen und gewährt nur Zugriff wenn das Programm als Root (uid=0) läuft, weshalb viele Programme mit Root-Rechten gestartet werden müssen (z.B. um einen CD-Brenner mit Echtzeit-Scheduling, einen Ping-Befehl auszuführen oder einen Web-Server auf Port 80 starten zu können). Dies geschieht per setuid Bit, d.h. das Programm kann dann sowohl als der Inhaber (=normalerweise Root) sowie als Benutzer laufen.
Es ergibt sich nun folgendes Problem: Besitzt das Programm eine Sicherheitslücke so ist es einem Benutzer ohne Root-Rechten möglich über das Programm Code mit Root-Rechten auszuführen. Da relativ viele Programme nur mit Root-Rechten ordnungsgemäß arbeiten können und einige davon auch relativ komplex und somit relativ anfällig für Sicherheitslücken

sind, ist dies ein sehr ernstzunehmendes Problem.

Ein anderer Ansatz ist die „System Privilege Table“, bei dem eine Liste mit Programmen, die spezielle Rechte haben müssen, gespeichert und vom Kernel abgefragt wird. Bei diesem Mechanismus kann die Datei allerdings selbst angegriffen und ausgetauscht werden und muß daher besonders vor Veränderungen geschützt werden. Sollte es allerdings einem Benutzer gelingen eine dieser Dateien auszutauschen oder zu ändern, so ist er ebenfalls in der Lage Code mit Root-Rechten auszuführen, wodurch ein schwer zu findendes Sicherheitsleck entstanden ist.

3. Access Control Listen (ACLs)

Bei den ACLs handelt es sich um eine Weiterentwicklung des normalen User-Group-Other Systems, sodaß die Einteilung feiner und spezifischer sein kann. Jedes Objekt hat dabei einen Liste in der die einzelnen Benutzer und Gruppen aufgeführt sind und die jeweiligen Aktionen die auf dieses Objekt zulässig sind. Dieses System eignet sich allerdings nur zur Beschränkung des Dateisystems, d.h. die restliche Zugriffsverwaltung ist damit nicht abgedeckt.

ACL-Systeme können den Zugriff verhindern und zurücknehmen, aber sie können weder den Zugriff limitieren noch ermöglichen. Alle Programme die unter meinem Login laufen können alle meine Daten bearbeiten oder löschen. Man kann versuchen dies folgendermaßen zu umgehen: neue ID für jede Arbeitsaufgabe. Allerdings bedeuten mehr IDs auch mehr Benutzungsaufwand, mehr Administrationsaufwand und längere ACLs an den Objekten, was letztendlich das Laufzeitverhalten sehr negativ beeinflussen. Je mehr Sicherheit man also will, umso mehr Aufwand hat man, was sich letztlich nicht rechnet. Es ist für mich auch nicht möglich einen Teil meiner Rechte weiterzugeben, solange ich nicht der Besitzer des Objekts bin (denn nur als Besitzer bzw. root kann ich die ACL verändern).

4. Capabilities

Der Begriff Capabilitiy geht auf Dennis und Van Horn zurück („Programming Semantics for Multiprogrammed Computations“, 1966). Die grundsätzliche Idee ist, daß ein Programm um auf ein Objekt zugreifen und bearbeiten zu können, einen speziellen Schlüssel benötigt. Dieser Schlüssel erlaubt es einen Menge von spezifischen Aktionen auf das Objekt durchzuführen und wird Capabilitiy genannt. Ein Benutzer (Subjekt) erhält eine Referenz auf ein Objekt, welche alle Rechte, die das Subjekt an dem Objekt besitzt enthält. Bei der Nutzung der Capability (Zugriff auf das Objekt) werden die Rechte überprüft und entschieden ob dieser gewährt wird oder nicht. Im Gegensatz zu den ACL-Listen, in denen steht welche Subjekte wie auf das Objekt zugreifen dürfen, gibt es die Capability-Listen oder kurz C-Listen, in denen die Capabilities gespeichert werden und somit indirekt auf welche Objekte das Subjekt zugreifen darf. Das System ist also vom Grundsatz her einschränkend, da „zunächst“ erst mal alles gesperrt ist und erst erlaubt werden muß, statt alles zu erlauben und im nachhinein zu sperren.

4.1 Eigenschaften von Capabilities

- Ein Capabilitiy kümmert sich nicht darum wer sein Besitzer ist. Einzig und allein der Besitz eines hinreichenden Capabilities gewährt den Zugriff, nicht etwa der Login/UID.
- Für ein Objekt kann es mehrer Capabilities geben, welche wiederum verschieden Zugriffsrechte gewähren können.

- Capabilities können den Zugriff auf eine Sammlung von anderen Capabilities ermöglichen.

- Capabilities können übertragen bzw. vererbt werden.
- Capabilities können kopiert werden.
- Capabilities können wieder „entzogen“ bzw. annuliert werden

4.2. Methoden zur Absicherung der Capabilities

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

- Tagged Capabilities:
  Spezielle Hardware stellt sicher, dass nur als Capabilitiy markierte Daten auch als Capabilities verwendet werden können. Auf Daten, die mit einem Capability-Bit gekennzeichnet sind, können dagegen keine arithmetischen Operationen ausgeführt werden, d.h. jede Speicherstelle und jedes Register muß ein solches "tag" Bit besitzen.
- Partitioned Capabilities:
  Capabilities und Daten werden getrennt voneinander gespeichert. Es gibt spezielle Capability Register, die nur aus speziellem Capability Speicher geladen werden können. Dies lässt 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 möglichen Capabilities sind gültig. Ein böswilliges Subjekt muss sehr viele Capabilities „würfeln“, bis eine gültige herrauskommt.

4.3 Funktionsweise Capability-Basierender Systeme

In einem solchen System hält jedes Programm einen Satz an Capabilities. Wenn jetzt ein Programm A ein Capabilitiy zur Kommunikation mit Programm B hat, dann können beide ihre Capabilities austauschen bzw. übertragen. Der einzige Weg um Capabilities zu erhalten führt normalerweise auch über irgend eine Art von Kommunikation.

Da eine unbegrenzte Anzahl an Capabilities das System mit der Zeit ausbremsen würde, ist meist jedem Programm nur eine begrenzte und kleine Anzahl (z.B. 16 oder 32) von Capabilities erlaubt (oft gibt es zusätzlich ein Mittel, um weitere Capabilities speichern zu können, falls diese unbedingt benötigt weren). Oberste Ziel ist es also, den Satz an Capabilities, der von jedem Programm gehalten wird, so spezifisch und klein wie möglich zu halten. Im Gegensatz zu den Superuser-Systemen kann man mit Capabilities das „Principle of last privilege“ realisieren, d.h. ein Programm darf gerade soviel im System machen, wie es ursprünglich vorgesehen ist, für weitere Aktionen besitzt es nicht die benötigten Capabilities bzw. Rechte.

Wenn in einem solchen System ein Programm eine Aktion auf ein Objekt ausführen will, dann muß es das Capability aufrufen und ihm die Art der Aktion übermitteln. Soll beispielsweise unter UNIX der Systemcall read(fd, buf, sz) aufgerufen werden, dann wird die Operation statt auf einen normalen Fildeskriptor auf ein Capabilitiy angewendet also read (cap, buf sz).

Ein Poblem bei Capabilities ist das Speichern ebendieser auf die Festplatte, sodaß man sie nach einem mehr oder weniger gewollten Neustart wieder zur Verfügung hat. Nimmt man beispielsweise an, daß unser Programm das Capabilitiy zum Erstellen einer Datei und zum Schreiben in ebendiese hatte. Alle unsere benötigten Informationen liegen jetzt in dieser Datei. Wenn es jetzt z.B. zu einem Stromausfall kommt haben wir nach dem Neustart folgendes Problem: Um auf die Datei zugreifen zu können benötigen wir zuerst ein Capabilitiy um auf das Dateisystem zugreifen zu können. Aber woher würden die ersten Programme ihre Capabilities beziehen und welche Autorität würde sie verteilen.

Eine einfache Lösung: Universal Persistence. Man merkt sich alles, d.h. man schreibt alle 5 Minuten alles auf, an dem der Computer arbeitet. Sollte es nun zu einem Ausfall kommen, dann arbeitet der Computer einfach an der zuletzt gespeicherten Stelle weiter und da dies auch alle laufenden Programme berücksichtigt, muß man sich nach dem Neustart nicht darum kümmern, wer auf was Rechte hat. Diese Lösung mag sich zwar ineffizient anhören, ist aber in der Realität schneller als andere Varianten und benötigt weniger Code.

5. Capabilities vs. Superuser

Nun zu den Vor- und Nachteile an einigen ausgewählten Beispielen:

5.1. Zugriffsbeschränkung (Confinement)

Angenommen man hat ein Programm das mit wichtigen geheimen Daten arbeitet. Man will das niemand ohne explizite Eigenwilligung auf diese Daten zugreifen kann, obwohl der Computer an einem Netzwerk hängt. Hat man in einem ACL-System Zugriff auf das Netzwerk, dann haben es die Programme unter meinem Login auch. In einem Capabilitiy-System gibt man dem Programm einfach kein Capabilitiy für den Netzwerkzugriff und schon hat das Programm keine Möglichkeit Daten über das Netzwerk zu verbreiten.

5.2. Priviligierte Programme

Das Programm zum ändern des Passworts benötigt die Rechte um die Passwortdatei sowohl lesen wie auch schreiben zu können, aber es darf diese nicht an weitere weitergeben. In einem Superuser-System gibt es nur einen Weg dies zu tun: Der Zugriff auf die Passwortdatei ist begrenzt auf einen speziellen Benutzer (der Superuser oder root) und es gibt Mittel um das Passwort-Programm als Superuser laufen zu lassen, wodurch sich die bereits genannten Sicherheitsprobleme ergeben. In einem Capability-System gewährt man einfach dem Passwort-Programm mit einem Capability Zugriff auf die Passwortdatei und gibt ein weiteres Capabilitiy an die Benutzer, daß das lediglich das Ausführen des Passwort-Programms erlaubt. Ein weiteres Beispiel wäre der Ping-Befehl, der ebenfalls von jedem ausführbar sein soll, jedoch Root-Rechte benötigt um einen Raw-Socket öffnen zu können. Man kann dem Ping-Programm einfach den Capabilitiy für das Senden roher IP-Pakete geben statt ihm gleich Root-Rechte zugestehen zu müssen.

5.3. Zusammenarbeit

Am schwierigsten ist der Umgang mit der gemeinsammen Nutzung eines Objekts. Angenommen man besitzt ein geheimes wertvolles Programm und hat Angst davor, daß jemand das Binary decompilieren und das Programm stehlen könnte. Jetzt will jemand das Programm benutzen um Daten auszuwerten, allerdings ohne diese zu zeigen. In einem Capabilitiy System ist es möglich, ein Programm laufen zu lassen, ohne das man den Code zu Gesicht bekommt. Gleichzeitig kann man aber auch die Rechte für das Programm so konfigurieren, daß es nicht möglich ist die fremden Daten einzusehen. Eine Möglichkeit dies in einem herkömmlichen oder ACL-System zu tun existiert nicht.

5.4. Die spezifische Rücknahme von Rechten

Der große Nachteil an einem Capabilitiy-System ist die Rechterücknahme, da die Schlüssel für ein Objekt im ganzen System verteilt sind und somit alle Objekte durchsucht werden müssten. Man könnte natürlich das „Schloß“ ändern, allerding würden dann auch sämtliche Programme und somit auch Benutzer die Rechte an diesem Objekt verlieren, was eine Neuverteilung von Capabilities erforderlich machen würde. Man könnte einem Benutzer auch wie im Superuser-System den Login löschen und evtl. neu einrichten, was allerdings auch

wieder einen gewissen Aufwand bedeuten würden.

6. Capabilities unter Linux

Seit der Version 2.1 des Linux Kernels arbeitet man an der Implementation von Capabilities, was ab der Version 2.2 schon halbwegs praxistauglich ist. Ziel war es die Root-Abhängigkeit diverser Programme abzuschaffen und die damit verbundenen Sicherheitslücken zu eleminieren. Capabilities helfen dabei, die Root-Rechte in kleinere Portionen zu packen. Im 2.2er Kernel wurden 7 der im Posix-Dokument 1003.1e herausgestellten sowie 20 weitere linuxspezifische Capabilities implementiert.

CAP_CHOWN	Erlaubt das Ändern der Dateiinhaberrechte
CAP_DAC_OVERRIDE	Setzt alle DAC Zugriffsbeschränkungen ausser Kraft
CAP_DAC_READ_SEARCH	Setzt alle DAC Zugriffsbeschränkungen die mit Lesen und Suchen zusammenhängen ausser Kraft
CAP_KILL	Erlaubt das Senden von Signalen an Prozesse anderer User
CAP_SETGID	Erlaubt das Ändern der GID
CAP_SETUID	Erlaubt das Ändern der UID
CAP_SETPCAP	Erlaubt das Übertragen und Entfernen von Capabilities
CAP_LINUX_IMMUTABLE	Erlaubt das verändern geschützter Dateien
CAP_NET_BIND_SERVICE	Erlaubt das Binden von Ports < 1024
CAP_NET_RAW	Erlaubt den Gebrauch von Raw-Sockets

7. Trusted Solaris: ACLs, Capabilities und administrative Rollen

Trusted Solaris verwendet eine ganze Sammlung von Sicherheitssystemen die inneinander greifen und sich gegenseitig ergänzen:

7.1 Mandatory Acces Controls (MAC)

Hierbei handelt es sich um ein System mit dessen Hilfe Daten mit „Labels“ in verschiedene hierarchische Sicherheitsstufen wie „public“, „private“, „private-engeneering“ eingeteilt werden können. Diese Einteilungen können von normalen Benutzern normalerweise nicht geändert werden und sorgt somit dafür, daß Informationen nicht „unabsichtig“ veröffentlicht oder ausgetauscht werden können.

7.2. Discretionary Access Controls (DAC)

DAC bedient sich der normalen Dateizugriffsrechte bzw. auf wunsch auch ACLs um den Zugriff auf Daten bzw. Informationen in Abhängigkeit der Benutzeridentität bzw. Gruppenmitgliedschaft zu beschränken. Im Gegensatz zum „normalen“ Solaris ist der Root nicht von diesen Beschränkungen ausgenommen. DAC wird zusammen mit MAC für die Kontrolle des Dateisystems verwendet.

7.3. Privileges (Capabilities)

In Trusted Solaris finden Capabilities ebenfalls Verwendung wodurch das „least privilege“-Prinzip umgesetzt werden kann. Die Implementierung unterscheidet sich im Detail zwar etwas von der im Linux Kernel oder der in anderen Capabilitiy-Systemen wie EROS, die grundsätzliche Funktionsweise bleibt jedoch die gleiche. Die Capabilities setzen Teile der DAC-Restriktionen ausser Kraft und Regeln hauptsächlich alle Sicherheits-beschränkungen die nicht mit Dateizugriff in Zusammenhang stehen.

7.4. Role-Based Access Control (RBAC)

Des weiteren werden unter Trusted Solaris die administrativen Teile auf mehrer Administratoren aufgeteilt, d.h. es gibt keinen allmächtigen Administrator mehr, sondern mehrere sich ergänzende Administratoren. Bevor die Administratoren mit der Arbeit beginnen können, müssen sie sich zunächst wie normale Benutzer im System einloggen und anschließend einen administrative Rolle übernehmen. Alle administrativen Aktivitäten können überwacht, protokolliert und zurückverfolgt werden, da sich die Administratoren vor der Übernahme authentifizieren müssen. Die administrativen Teile werden standardmäßig in folgende Rollen gesplittet: Security administrator (secadmin), System administrator (admin), Operator (oper)

7.5. Rights Profiles

Trusted Solaris enthält eine Datenbank mit funktionsorientierten Rechte-Profilen, die in hierarchischer Form miteinander kombiniert oder abgeändert werden können. Mit Hilfe dieser Profile kann man schnell und effektiv die Rechte und Restriktionen für spezifische Benutzer, Gruppen oder Rollen verteilen.

8. Fazit

Wenn Capabilities wesentlich besser wie herkömmliche Superuser-Systeme sind, warum werden sie dann nur recht selten verwendet? Nun zum einen gibt es neben den vielen Vorteilen auch Nachteile bei Dingen wie etwa der Rechterücknahme oder der Rechtekonsistenzhaltung beim Systemneustart, zum anderer Teil des Problems ist historisch bedingt. Die ersten Capability-Systeme wurde in Hardware gebaut und sollten den Zugriff auf den Hauptspeicher regeln. Das machte sie extrem langsam und komplex und führte zu einem äußerst schlechten Ruf. Auch in den 80ern wurde nur recht schlechte Arbeit an Microkernels (die den Capability-Systemen ähnlich sind) abgeliefert und ebenso schlechte Analysen ergaben, daß das ein Problem der Microkernels im allgemeinen sei (obwohl es an schlechten Implementationen lag).

Des weiteren lohnen sich derartige Systeme auch erst ab einer gewissen Größe, d.h. wenn ein kleines System mit nur ein oder zwei Administratoren und einer überschaubaren Anzahl an Benutzern zu administrieren ist und dabei die Sicherheit des Systems nicht die entscheidende Rolle spielt, so kann man dies auch mit einem normalen Superuser-System problemlos angehen.

Übrigens gibt es auch UNIX-kompatible Umgebungen, die auf ein Capability-System aufsetzen, wenngleich dies andersherum nicht möglich ist. Existierender Code kann also trotzdem weiterverwendet werden. Wer interesse hat kann sich entweder mit dem EROS-System oder dem RSBAC-Project beschäftigen (siehe Quellen).

9. Quellen

[1] Linuxsecurity.com: Linux 2.4 - Next Generation Kernel Security
http://www.linuxsecurity.com/feature_stories/kernel-24-security.html

[2] Linux Capabilities FAQ 0.2
ftp://ftp.guardian.no/pub/free/linux/capabilities/capfaq.txt

[3] SecurityFocus HOME Infocus: Introduction to Linux Capabilities and ACL's
http://www.securityfocus.com/infocus/1400

[4] Skyhunter.com: Introduction To Capability Based Security
http://www.skyhunter.com/marcs/capabilityIntro/index.html

[5] EROS-OS.org: Essays on Capabilities and Security
http://www.eros-os.org/essays/00Essays.html

[6] Norman Hardy's Homepage
http://www.cap-lore.com/index.html

[7] Trusted Solaris Operating System
http://wwws.sun.com/software/solaris/trustedsolaris/index.html

[8] Rule Set Based Access Control (RSBAC) for Linux – Overview
http://www.rsbac.org

[9] Ievlev Stanislav's Homepage
http://linux.ru.net/~inger/

[10] Betriebssystem HOWTO für UNIX/Linux von Prof. Jürgen Plate
http://www.fs.ei.tum.de/admin/howto/unix/