Kapselung mobiler Programme

Lars Reuther, Hermann Härtig
Technische Universität Dresden
Lehrstuhl für Betriebssysteme
{reuther,haertig}@os.inf.tu-dresden.de

Zusammenfassung

Dieses Papier beschreibt eine Technik zur effektiven Isolierung von Prozessen, die mobile Programme ausführen. Sie basiert darauf, daß Programme in Form einer *Wunsch-Liste (wish list)* vor ihrer Ausführung kundgeben, welche Betriebsmittel sie zur Erledigung der versprochenen Funktionalität benötigen. Vor dem Start eines Programms wird interaktiv und anhand einer *Vertrauens-Liste (trust list)* überprüft, ob dem Programm in Bezug auf die angeforderten Betriebsmittel auf dem System Vertrauen entgegengebracht wird. Bei Programmstart entsteht somit eine *Kann-Liste (capability list)*, die alle dem Programm zugänglichen Betriebsmittel enthält und deren Einhaltung durch die Umgebung des ausführenden Prozesses durchgesetzt wird. Die angesprochenen Listen enthalten symbolische Elemente in einer Form, die eine Anwendung dieser Technik an unterschiedlichen Stellen einer einzelnen Systemarchitektur ebenso wie in heterogenen Systemen zuläßt. Das Papier beschreibt und begründet die Technik, berichtet über eine Integration in ein Mikrokern-basiertes UNIX System und präsentiert die Ergebnisse von Leistungsmessungen. Abschließend wird über die zusätzlichen Probleme berichtet, die bei der Übertragung dieser Technik auf mobile Objekte, z.B. Mobile Agenten, entstehen.

1 Einführung

Jüngste Entwicklungen machen deutlich, daß freizügiges Umhersenden ausführbarer Programme immer größere Bedeutung für arbeitsteilige Systemarchitekturen erlangt. Offensichtlich wird dies besonders bei den sogenannten *Applets*, also Programmen, die von Servern auf Arbeitsstationen geladen und dort ausgeführt werden. Auch die sogenannten *Netzwerkcomputer* basieren darauf, daß Programme nicht mehr lokal gehalten, sondern grundsätzlich über das jeweilige Netz bezogen werden.

Applets und Netzwerkcomputer stellen jedoch aus technischer Sicht nur die Spitze eines Eisbergs dar. In verschiedenen internationalen Forschungsprojekten wird an Erweiterungen dieser Technik gearbeitet: Unter *Servlets* versteht man Programme, die von Klienstationen zu Servern versandt und dort ausgeführt werden. *Mobile Agents* [1] sind mobile Programme, die in Netzen umherwandern und Stellvertreteraufgaben für ihre Besitzer übernehmen. *Active Networks* [2] basieren auf der Idee, Netzwerkprotokolle nicht statisch festzulegen; neben Datenpaketen können auch Programmpakete verschickt und auf den Netzwerkrechnern (z.B. Routern) ausgeführt werden, was zu einer außerordentlichen Flexibilität führt. Insgesamt läßt sich nicht ausschließen, daß hier ein tiefgreifender Paradigmenwandel für den Aufbau von Systemen im Gange ist, dessen Auswirkungen noch nicht absehbar sind.

Offensichtlich sind jedoch die Risiken, die mit dem Einsatz einer solchen Technik einhergehen, z.B. Trojanische Pferde.

Abschnitt 2 dieses Papiers diskutiert die Einsetzbarkeit gegenwärtig üblicher Schutzmechanismen für derartige Umgebungen. Insbesondere wird deutlich, wie durch das heute übliche Geschlossenhalten (beispielsweise durch Restriktion auf bestimmte virtuelle Maschinen) die Eleganz und Einsatzbreite der mobilen Programme verlorengehen. Abschnitt 3 beschreibt eine einfache, aber effektive Technik zum Umgang mit diesen Risiken. Sie basiert auf der Adaption von Techniken, wie sie schon in älteren Forschungsbetriebssystemen eingesetzt wurden (etwa in der BirliX Security Architecture [3]). Der dann folgende Abschnitt beschreibt eine Implementierung des Konzepts, die die Autoren in einer Mikrokern-basierten Linux-Umgebung vorgenommen haben und nennt die resultierenden Leistungseckdaten. Eine Diskussion einiger weitergehender Probleme beschließt das Papier.

2 Derzeit eingesetzte Techniken

Der vielleicht bekannteste und dennoch kaum eingesetzte Ansatz zum Umgang mit potentiellen Trojanischen Pferden sind regelbasierte Sicherheitssysteme (*mandatory access control policies*), wie z.B. das durch Bell LaPadula formalisierte Modell [4] in seiner Multics-Interpretation. Es beschränkt den Fluß von Informationen auf der Basis einiger Regeln, deren Einhaltung das zugrundeliegende System durchsetzen muß. In der Praxis zeigte sich jedoch, daß solche Systeme sehr schwer zu administrieren sind. Der Effekt der Informationsaufwertung (*information upgrade effect*) ist wohlbekannt. Hinzu kommt, daß bislang relativ wenig solcher regelbasierten Systeme bekannt sind, deren praktische Einsetzbarkeit — nach Ansicht der Autoren — auch deutlich limitiert ist, insbesondere wenn man an das o.g. Szenario denkt, in dem Programme in globalen Netzwerken „umherwandern".

Mit *Firewalls* können lokale Rechnernetze von einem globalen Netz teilweise isoliert werden, z.B. durch den Einsatz von Paketfiltern oder Proxies [5]. Für den Einsatz mobiler Programme wird jedoch vielmehr eine Technik benötigt, die nicht Systeme, sondern Prozesse oder Prozeßmengen innerhalb von Systemen isoliert, also gewissenmaßen Firewalls innerhalb des Systems.

Zugriffssteuerlisten (z.B. rudimentär in UNIX-Systemen) sind ein weitverbreiteter Ansatz zum Schutz von Daten auf lokalen Rechnersystemen. Zum Umgang mit mobilen Programmen könnten etwa für jedes mobile Programm (bzw. eine Gruppe von mobilen Programmen) ein neuer Nutzer bzw. eine neue Nutzergruppe eingeführt und für diese die Zugriffsrechte entsprechend gesetzt werden. Eine große Anzahl mobiler Programme in einem System führt jedoch schnell zu einer unübersichtlichen Verteilung der Zugriffsrechte, so daß eine sichere Verwaltung erschwert wird bzw. unmöglich ist, mit anderen Worten: Zugriffssteuerlisten sind nicht skalierbar. Allgemein reichen Zugriffssteuerlisten für die Durchsetzung des Prinzips der geringstmöglichen Privilegierung (*least privilege*) bekanntlich nicht aus [3].

In heute üblichen Systemen, die mobile Programme nutzen, werden sehr harte interne Firewalls, *Sandboxes* genannt, eingesetzt. Beispielsweise wird von einigen Java-Interpretern schlicht der Zugriff auf Dateien unterbunden (*sandboxing*). Es wird sofort deutlich, daß sich damit der Anwendungsbereich für Applets sehr stark einschränkt. Ganz indiskutabel wird der Ansatz bei weitergehenden Anwendungen von Applets oder allgemeiner den mobilen Programmen. Eine *data mining* Anwendung beispielsweise ist dann völlig ausgeschlossen.

Ein anderer Ansatz schlägt vor, mobile Programme von ihren Autoren oder Anbietern signieren zu lassen, so daß im Schadensfall die Autoren zur Rechenschaft gezogen werden können. Aber auch hier werden die Grenzen rasch deutlich. Ein einigermaßen clever angerichteter Schaden kann erst lange nach der Ausführung eines mobilen Programmes oder — bei Vertraulichkeitsverletzungen — gar nicht sichtbar und damit zuordenbar werden.

Manche Betriebssysteme — die Pioniersysteme sind Hydra [6], CAP [7] und Amoeba [8] — nutzen *Capability-Listen* für die Kapselung von Prozessen. Diese enthalten alle Objekte, auf die ein Prozeß im Moment zugreifen darf. In der BirliX-Sicherheitsarchitektur [3] erhalten suspekte Programme sogenannte *Subjektrestriktionen*. Sowohl Programme als auch aktive Prozesse nehmen ihre Subjektrestriktionen mit, wohin immer sie migrieren. Diese Mechanismen lassen zwar eine sehr feingranulare Rechtevergabe nach dem Prinzip der geringstmöglichen Rechteausstattung (*least privilege*) zu, besitzen aber einen elementaren Nachteil; sie sind nur in homogenen Systemen einsetzbar und somit für die Anwendung in oben skizzierten Umgebungen ungeeignet.

Aus der Diskussion der aktuellen Verfahren ergeben sich folgende Anforderungen an ein System zur sicheren Ausführung mobiler Programme:

- Prozesse, die mobile Programme ausführen, müssen effektiv isoliert werden können.

- Trotz der Isolation müssen Zugriffe auf Systemressourcen erlaubt sein, die für das Funktionieren des Programms benötigt werden.

- Der Eigentümer/Administrator eines Rechnersystems soll in der Lage sein, Vertrauen oder Mißtrauen in Programme und deren Hersteller auszudrücken.

- Das System muß einfach administrierbar sein.

- Das System muß sich auch in Umgebungen mit einer theoretisch unbegrenzten Anzahl an mobilen Programmen einsetzen lassen.

- Die Technik sollte auch in heterogenen Systemen anwendbar sein.

Im folgenden Abschnitt wird eine Adaption der BirliX-Subjektrestriktionen beschrieben, die das Potential für die Erfüllung dieser Anforderungen besitzt.

3 Die *-Listen

Die prinzipielle Idee soll anhand der allgemeinen, d.h. in verschiedensten Umgebungen einsetzbaren Vorgehensweise der teilnehmenden Parteien erläutert werden. Eine Konkretisierung und Implementierung auf ein Linux-System wird später beschrieben.

Zunächst fügt der Hersteller oder Anbieter eines Programms diesem eine Liste symbolischer Namen bei. Diese Liste enthält die Namen aller Betriebsmittel (Dateien, Netzwerkverbindungen, Prozesse, ...), welche das Programm für die Erbringung der ausgewiesenen Funktionalität benötigt. Zusätzlich gibt der Hersteller seinem Programm einen Namen. Das Programm wird zusammen mit dieser Liste und dem Namen durch den Hersteller signiert (wir gehen davon aus, daß es einen sicheren und effektiven Weg zur Überprüfung dieser Signatur gibt, z.B. mittels zertifizierter öffentlicher Schlüssel). Das Ergebnis dieses Schrittes ist ein Programm, erweitert um eine Liste der benötigten Betriebsmittel. Diese Liste nennen wir im folgenden die *Wunsch-Liste (wish list)* eines Programms.

Der Administrator (bzw. Eigentümer) eines Rechnersystems bestimmt, in welchem Umfang er einem Programm bzw. dessen Hersteller vertraut. Dies geschieht durch die Angabe aller Betriebsmittel, für die Programme eines bestimmten Herstellers Zugriffsrechte besitzen sollen. So kann Programmen des Herstellers Foo-soft Zugriff auf alle Dateien des

Verzeichnisses `/windows` erlaubt werden, während Shareware Programmen nur Zugriffe auf Dateien im Verzeichnis `/tmp` gestattet sein sollen. Darüber hinaus können bestimmten Programmen mehr Rechte eingeräumt werden, als es durch die seinem Hersteller eingeräumten Rechte besitzen würde. Ergebnis dieses Schritts ist also eine Datenbank, die Herstellern und Programmen Ressourcen zuordnet. Im folgenden wird diese Datenbank *Vertrauens-Liste (trust list)* genannt.

Soll nun ein mobiles Programm ausgeführt werden, werden als erstes der Hersteller des Programms ermittelt sowie die Signatur des Programms und die dazugehörende Wunsch-Liste überprüft. Dann wird getestet, ob die zu dem Programm gehörende Wunsch-Liste eine Teilmenge der zu dem Hersteller bzw. Programm gehörenden Vertrauens-Liste ist. Ist dies der Fall, kann das Programm sofort ausgeführt werden, anderenfalls wird dazu eine interaktive Komponente, im weiteren als *Supervisor* bezeichnet, zur Zulässigkeit des Zugriffs befragt. Das Ergebnis dieses Schrittes, also die Menge von Ressourcen, auf die der das mobile Programm ausführende Prozeß zugreifen darf, wird im folgenden *Kann-Liste (capability list)* genannt und stellt eine Variante der Eingangs genannten Capability-Listen dar.

Während der Abarbeitung des Programms wird die Kann-Liste (wie klassische Capability-Listen) zur Überprüfung von Zugriffen auf Ressourcen verwendet. Es werden nur Zugriffe erlaubt, für die ein entsprechender Eintrag in der Kann-Liste existiert. Bei Zugriffen auf Ressourcen, für die kein solcher Eintrag vorliegt, wird eine entsprechende Anfrage an den Supervisor gesendet. Dieser kann den Zugriff erlauben, ablehnen oder die Ressource transparent für das Programm durch eine andere ersetzen.

Startet ein Programm ein weiteres, wird dieses auf die gleiche Weise überprüft. Die neue Kann-Liste wird gebildet, indem die Schnittmenge der Kann-Liste des aktiven Programms, der Vertrauens-Liste und der Wunsch-Liste des neuen Programms gebildet wird. Dadurch wird erreicht, daß ein Programm durch Ausführen eines anderen Programms keine zusätzlichen Rechte erlangen kann; die Zugriffsrechte können so nur geringer werden. Diese Festlegung wird später noch durch ein Beispiel aus dem Linux-Kontext näher erläutert.

Die eben beschriebenen Listen werden im folgenden allgemein mit **-Listen* bezeichnet.

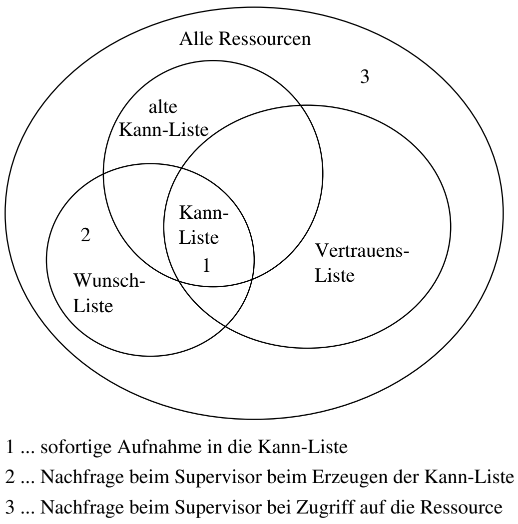

Abbildung 1: *-Listen

Einträge in den *-Listen sind symbolische Namen, welche auch Wildcards[1] enthalten können.

Abb. 2 zeigt ein Beispiel für die Anwendung der *-Listen. Die Vertrauens-Liste für den Hersteller Foo-soft erlaubt den Zugriff auf alle Dateien ab dem Verzeichnis `/pub/docs`. Die Wunsch-Liste des Programms verlangt den Zugriff auf alle Dateien ab `/pub/docs/techreports`, demzufolge wird `/pub/docs/techreports+` in die Kann-Liste aufgenommen.

Vertrauens-Listen dürfen auch Alias-Einträge enthalten, die zur transparenten Ersetzung von Ressourcen verwendet werden

[1] In der derzeitigen Implementierung werden zwei Wildcards verwendet. + erlaubt den Zugriff auf alle Dateien und Verzeichnisse unterhalb des angegebenen Verzeichnisses, * nur den Zugriff auf Dateien innerhalb dieses Verzeichnisses.

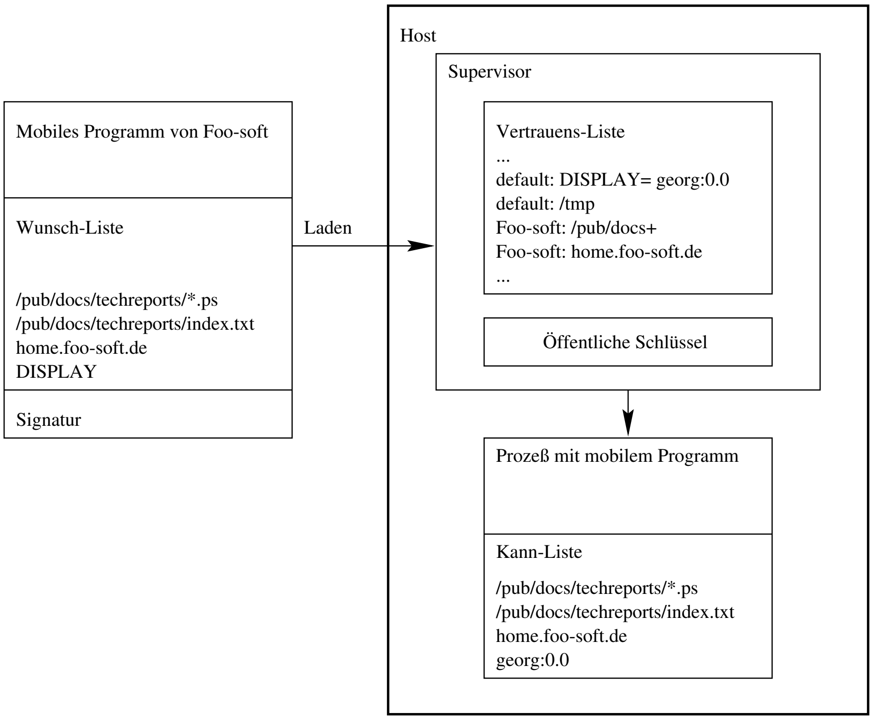

Abbildung 2: Ein Beispiel für *-Listen

können. Beispielsweise enthält die Wunsch-Liste aus Abb. 2 den Alias-Namen `DISPLAY`. Dieser Alias-Name wird durch den Eintrag in der Vertrauens-Liste in den Namen `georg:0.0` aufgelöst.

Ein häufig an dieser Stelle der Diskussion vorgebrachter Einwand lautet, warum denn eine Vertrauens-Liste benötigt werde, wenn ohnehin ein Supervisor angefragt werden könne. Der Grund liegt darin, daß solche Anfragen minimiert werden sollen, da sonst durch Gewohnheit bei der Beantwortung dieser Anfragen Nachlässigkeit entsteht. Ein anderer Einwand betrifft die Notwendigkeit der Wunsch-Liste, da doch in der Vertrauens-Liste ohnehin alles festgelegt sei, was ein Programm benutzen dürfe. Die Wunsch-Liste soll sicherstellen, daß ein Hersteller alle Betriebsmittel anzugeben hat, die sein Programm benötigt, so daß am Anfang die für die Ablauffähigkeit des Programms notwendigen Betriebsmittel geprüft werden können. Anderenfalls könnte eine Situation entstehen, in der lange nach Start eines Programms die weitere Erbringung der Funktionalität vom Zugriff auf weitere Betriebsmittel abhängig gemacht wird.

Eine weitere Frage bezieht sich auf die Wechselwirkung mit anderen Schutzmechanismen, etwa Zugriffssteuerlisten. Grundsätzlich dienen *-Listen zur *Einschränkung* von Rechten mobiler Programme. Das heißt, daß etwa Zugriffssteuerlisten grundsätzlich zusätzlich überprüft werden.

Zuletzt soll hier die Aufnahme spezifischer Operationen in die *-Listen diskutiert werden. Es wäre etwa in einem UNIX-Kontext denkbar, daß Vertrauen in einen Hersteller gesetzt wird, eine Menge von Dateien zu lesen, aber nicht zu modifizieren. Die von den Autoren im Linux-Kontext vorgenommene Implementierung besitzt diese Möglichkeit. Indes ist in einem weiteren Kontext — etwa in der Umgebung eines botschaftenbasierten Objektsystems — nicht klar, wie durch einen Abfangmechanismus festgestellt werden soll, ob eine Botschaft keine modifizierende Wirkung hat und damit zugestellt werden darf oder nicht.

Die hier beschriebene Technik kann an verschiedenen Punkten einer Systemarchitektur integriert werden. Eine offensichtliche und einfache Möglichkeit ist die Integration in Interpreter bzw. Compiler für Sprachen oder Virtuelle Maschinen, die Mobilität von Programmen unterstützen (z.B. eine Implementation der *Java Virtual Machine*). Dieser Ansatz bietet zwar die Möglichkeit einer einfachen Portierung, hat aber zwei wesentliche Nachteile:

- er beschränkt die Anzahl der Programme auf die, die in einer solchen Sprache programmiert sind, verletzt also das Prinzip der Offenheit und
- für jede Programmiersprache muß die Technik neu implementiert werden.

Eine Integration in das Betriebssystem (speziell in die Systemruf-Schnittstelle) besitzt diese Nachteile hingegen nicht. Für diese Umsetzung werden Hardwarekonzepte (Schutz von Adreßräumen) und Software (der eingefügten Funktion in die Systemruf-Schnittstelle) benutzt. Dies wurde erfolgreich an einem Prototyp auf Basis des Linux-Systems durchgeführt.

4 Implementierung

Für die Umsetzung der *-Listen in einem Betriebssystem muß dieses über einen Mechanismus zur Isolation von Prozessen und die Möglichkeit zum Unterbrechen eines Systemrufs verfügen. Im folgenden wird die Implementierung der *-Listen in eine Mikrokern-basierte UNIX-Umgebung beschrieben.

4.1 Allgemeine Prinzipien

Bild 3 zeigt die Prozeßstruktur eines Systems zur Ausführung von mobilen Programmen.

Zunächst wird der Supervisorprozeß gestartet, der durch einen neu eingeführten Systemruf bewirkt, daß alle Abkömmlinge (Kind-Prozesse) mit Kann-Listen ausgestattet werden und daß grundsätzlich die relevanten Systemaufrufe abgefangen und die Kann-Liste inspiziert werden. Zusätzlich wird dadurch der

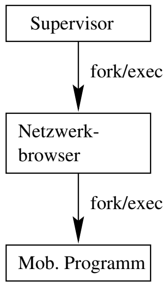

Abbildung 3: Prozeßstruktur für mobile Programme

aufrufende Prozeß als zuständig für die erforderlichen Rückfragen und die Modifikation der Kann-Liste aller Abkömmlinge verbindlich festgelegt. Danach kann durch Aufrufe von `fork` und `exec` ein Programm, das mit mobilen Programmen umgeht (z.B. Netzwerkbrowser) gestartet werden. Vorteilhaft an dieser Konstruktion ist, daß die Anfragen und die Verwaltung der Kann-Liste vom Supervisor — und zwar transparent für den Netzwerkbrowser — vorgenommen wird, so daß solche Programme ohne Modifikationen übernommen werden können.

Bild 4a zeigt nun eine Situation, welche die früher beschriebenen Festlegungen unterstreichen soll, daß grundsätzlich Kann-Listen bei Start eines neuen Programms nur weiter eingeschränkt werden.

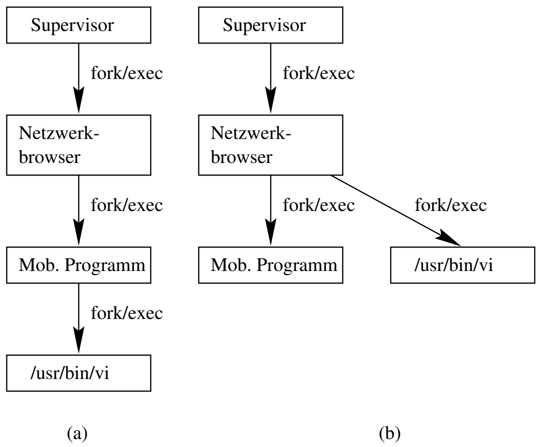

Abbildung 4: Weitergabe der Kann-Listen

In dem Beispiel wird aus einem mobilen Programm heraus ein Editor aufgerufen, für den naturgemäß ein liberaler Eintrag in der Vertrauens-Liste vorliegt. Auf diese Weise könnte — falls nicht die Kann-Liste des aufrufenden Prozesses übernommen würde — durch geschickte Parameterwahl Zugriff auf Ressourcen erfolgen, die nicht in der Kann-Liste des mobilen Programms enthalten sind. Falls dennoch der Editor mit größeren Rechten benötigt wird, kann dies durch eine entsprechend andere Prozeßstruktur erfolgen (Abb. 4b).

4.2 Linux/L4

Als Grundlage für die Implementierung dient eine Portierung des Betriebssystems Linux auf den Mikrokern L4 [9]. Abbildung 5 zeigt die allgemeine Struktur dieses Ports. Ein Linux-Prozeß wird durch zwei L4-Tasks dargestellt. In der Nutzertask wird die Anwendung ausgeführt, die Kerntask stellt die Funktionalität des Linux-Kerns zur Verfügung. Systemrufe der Anwendung werden auf L4-Interprozeßkommunikation (IPC) zwischen der Nutzer- und der Kerntask abgebildet.

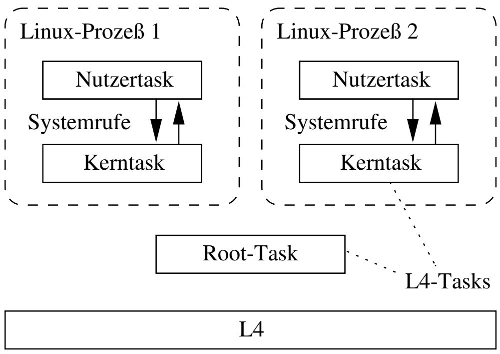

Abbildung 5: Linux/L4

4.3 Clans & Chiefs

Der Mikrokern L4 bietet eine elegante Möglichkeit zum Abfangen von Nachrichten zwischen Tasks. Tasks können zu einem *Clan* zusammengefaßt werden. Jeder Clan besitzt einen *Chief*. Wird bei einer Kommunikation zwischen zwei Tasks eine Clangrenze überschritten, wird die Nachricht durch L4 zu dem zugehörigen Chief umgeleitet. Dieser kann sie inspizieren, modifizieren, weiterleiten, umleiten oder unterdrücken (siehe Abb. 6) [10].

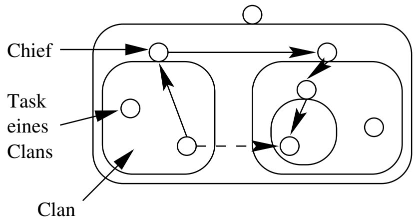

Abbildung 6: Clans & Chiefs

4.4 Struktur

Die Implementierung der *-Listen in Linux/L4 benutzt den Clans & Chiefs-Mechanismus zum Abfangen der Nachrichten zwischen der Nutzer- und Kerntask eines Linux-Prozesses. Abbildung 7 zeigt die Struktur der Implementierung. Dabei wird der Nutzerprozeß, welcher das zu überwachende Programm ausführt, durch einen Chiefprozeß kontrolliert. Die Nutzertask des Chiefprozeß ist dabei Chief des Clans, in dem die Nutzertask des Nutzerprozeß liegt. Da die Systemrufe der Anwendung auf IPC zwischen der Nutzer- und Kerntask abgebildet und diese über den Chief umgeleitet werden, hat der Chiefprozeß Kontrolle über alle Systemrufe der Anwendung.

Um die Argumente eines Systemrufs auswerten zu können, benötigt der Chiefprozeß Zugriff auf den Adreßraum des Anwendungsprogramms. Dies wird realisiert, indem die entsprechenden Speicherseiten aus dem Anwendungsprogramm in den Adreßraum der Chieftask eingeblendet werden. Die Ersetzung von Argumenten eines Systemrufes erfolgt, indem vor der Weitergabe der IPC-Nachricht an die Kerntask des Anwendungsprogramms die Argumente der Nachricht manipuliert werden.

4.5 Überwachte Ressourcen

Die derzeitige Implementierung betrachtet zwei Arten von Ressourcen: Dateien und Netzwerkverbindungen. Die Bezeichnung von

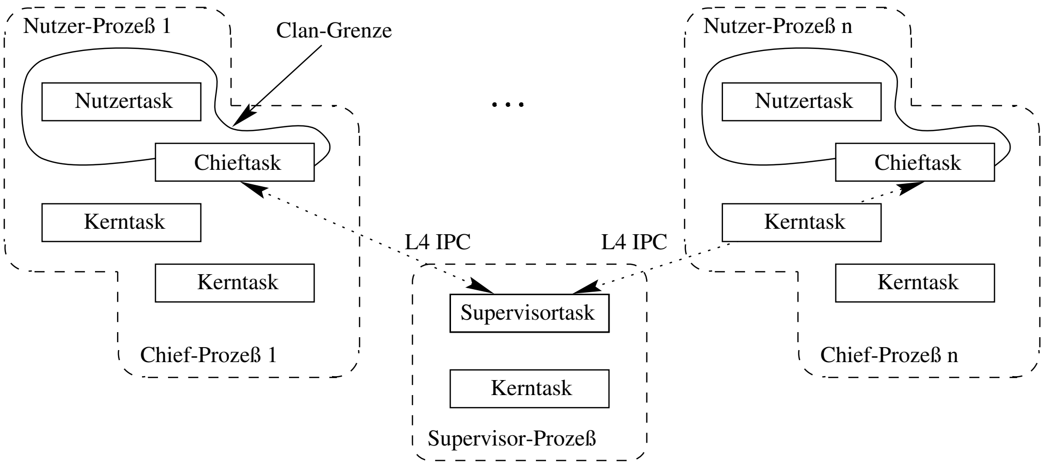

Abbildung 7: Taskstruktur

Dateien erfolgt über symbolische Dateinamen; Netzwerkverbindungen werden durch IP-Adressen und Port-Nummern beschrieben.

Um Zugriffe auf diese Ressourcen zu kontrollieren, muß nur ein geringer Teil der Systemrufe der Anwendung (ca. 20% der Systemrufe des Linux-Kerns) abgefangen und ausgewertet werden. Zu diesen Systemrufen gehört der `open`-Systemruf zum Öffnen einer Datei, hingegen müssen `read` und `write` Systemrufe nicht ausgewertet werden, da sie zum Zugriff auf die Datei einen Deskriptor verwenden, der durch `open` erzeugt wird, die Kontrolle der Zugriffsrechte demzufolge bereits erfolgt ist.

Für die Kontrolle der Zugriffe auf Netzwerkverbindungen sind u.a. die Systemrufe `connect`, `bind` und `accept` auszuwerten, die Sende- und Empfangsoperationen brauchen analog zu den Schreib-/Leseoperationen auf Dateien nicht ausgewertet zu werden, denn diese verwenden durch `connect` bzw. `accept` erzeugte Socket-Deskriptoren.

Erzeugt ein überwachter Prozeß mittels eines `fork`-Systemrufs einen Kind-Prozeß, wird diesem die Kann-Liste des Eltern-Prozeß zugewiesen, wodurch die Rechteweitergabe realisiert wird.

Wird ein neues Programm durch einen `exec`-Systemruf ausgeführt, so wird eine neue Kann-Liste für den Prozeß entsprechend der bereits beschriebenen Vorgehensweise bestimmt.

4.6 Probleme

Während der Implementierung wurden einige Probleme deutlich, die aus dem Zusammenwirken der *-Listen mit anderen Mechanismen des Betriebssystems resultieren.

Symbolische Links des UNIX-Dateisystems erschweren die Auswertung der Dateinamen eines `open`-Systemrufs deutlich. Um die korrekte Abbildung eines Namens in der Kann-Liste auf die Datei im Dateisystem zu gewährleisten, muß der Dateiname vollständig ausgewertet werden (d.h. die Pfadangabe muß auf evtl. symbolische Links untersucht werden), ein einfacher Vergleich der Namen ist nicht ausreichend.

Besondere Aufmerksamkeit erfordert auch die Verwendung von *setuid*-Programmen und der Möglichkeit des Ersetzens von Ressourcen. Enthält eine Vertrauens-Liste ein Alias auf die Passwortdatei `/etc/passwd`, kann ein Prozeß Root-Rechte erlangen, indem er in dieser Datei einen entsprechenden Eintrag einfügt und z.B. `su` ausführt. Um dies zu verhindern, wird in der derzeitigen Implementierung ein *setuid*-Programm nur dann in einem überwachten Prozeß ausgeführt, wenn der zugehörige Chiefprozeß über die entsprechenden Rechte verfügt. Das gerade erwähnte `su` wird demzufolge nur noch dann ausgeführt, wenn der Chiefprozeß Root-Rechte besitzt.

4.7 Messungen

Für die Leistungsbewertung wurde für einige Systemrufe die Zeit bestimmt, die bei der Bearbeitung zusätzlich durch die Auswertung der Kann-Liste entsteht. Tabelle 1 stellt die Ergebnisse dieser Messung auf einem Pentium/133 dar.

	Linux/L4	
Systemruf	ohne *-Listen	mit *-Listen
`getpid`	13 μs	23 μs
`open`	141 μs	274 μs
`connect`	223 μs	341 μs

Tabelle 1: Meßergebnisse

Die Werte für den `getpid`-Systemruf verdeutlichen die Zeit, die durch das Umleiten der IPC-Nachricht über den Chiefprozeß entstehen (10 μs). Die Werte für den `open`- und `connect`-Systemruf stellen den Aufwand für die Überprüfung eines Dateinamens bzw. einer IP-Adresse dar. Der hohe Aufwand für die Auswertung einer IP-Adresse (118 μs für die Auswertung eines 32-Bit-Wertes im Vergleich zu 133 μs für die Behandlung eines Dateinamens) resultiert aus dem Aufbau der Argumente des `connect`-Systemrufes. Der Zugriff auf diese Argumente erfordert zwei Zugriffe auf den Adreßraum des Anwendungsprogramms, während für die Auswertung eines `open`-Systemrufs nur ein Zugriff notwendig ist.

5 Weiterführende Arbeiten

Einige offene Probleme bzw. notwendige Erweiterungen der vorgestellten Technik sollen hier noch angesprochen werden.

Bei der Integration in Linux/L4 sowie bei allen Beispielen wurden als Betriebsmittel *logische Betriebsmittel*, d.h. Dateien, Verbindungen u.ä. angenommen. Tatsächlich ist jedoch auch die Überwachung physischer Betriebsmittel wie Speicher, CPU-Zeit und Bandbreite von hoher Bedeutung. Während es relativ einfach erscheint, maximalen Speicherverbrauch in *-Listen zu beschreiben und dies auch zu überwachen, ist die Überwachung für CPU-Zeit und Bandbreiten deutlich schwieriger, da die gängigen Betriebssysteme über solche Überwachungsmöglichkeiten nicht verfügen. An der TU Dresden wird z.Z. versucht, Betriebsmittelreservierungtechniken, die in Zusammenhang mit QoS-Architekturen untersucht und realisiert werden, für solche Zwecke einzusetzen.

Mobile Objekte bringen einen zusätzlichen Aspekt ins Spiel, besonders dann, wenn sie als migrierende Prozesse implementiert werden. Es können zwar wie gehabt Programme und ihre Wunsch-Listen per Signatur geschützt werden, aber Rechnerbetreiber müssen nun weitere Parteien zu einem gewissen Grad in ihr Vertrauen einbeziehen, nämlich diejenigen Rechner (bzw. deren Betreiber), die von den migrierenden Prozessen schon besucht wurden. Diese können an den Daten der Prozesse, also beispielsweise an Systemaufrufparametern Modifikationen vornehmen. Ein so möglicher Angriff soll kurz skizziert werden. Eine Durchgangsstation eines migrierenden Prozesses manipuliert den Stack so, daß bei dem Rücksprung aus einer Prozedur nicht in das eigentliche Programm, sondern in neu eingefügten Code auf dem Stack oder im Datenbereich gesprungen wird. Es stellt sich die Frage, ob auch Datenbereiche durch Signaturmechanismen geprüft werden sollten und ob die besuchten Rechner mit zu dem Vertrauensbereich gehören.

Die vorgestellte Technik basiert auf symbolischen Listen. In Anwendungen, die hohe Effizienzanforderungen haben, also etwa die eingangs angesprochenen aktiven Netzwerke, muß wohl auf eine andere Form der Benennung übergegangen werden.

6 Verwandte Arbeiten

In einem ersten Ansatz wurden die *-Listen direkt in den Linux-Kern integriert [11]. Einige der dabei aufgetretenen Probleme konnten in der in dieser Arbeit beschriebenen Implementierung gelöst werden.

An der University of California, Berkeley wurde ein System zur Überwachung der Systemrufe unter Verwendung der Solaris Prozeßsystem-Schnittstelle implementiert [12]. Die Auswertung der Systemrufe erfolgt über *Policy Modules*, die über Konfigurationsdateien eingerichtet werden. Ein Einsatz für die Behandlung von mobilen Programmen ist daher nur bedingt möglich.

Das von IBM entwickelte *FlexxGuard*-System verwendet eine Technik ähnlich der *-Listen zur sicheren Ausführung von Java-Applets. Im Unterschied zu den *-Listen wird die Zugriffskontrolle in der Ausführungsumgebung der Applets durchgeführt. Die Kontrolle bleibt daher auf Java-Applets beschränkt, ein von einem Applet gestartetes Programm kann nicht kontrolliert werden.

7 Zusammenfassung

In diesem Papier wird eine Technik zur effektiven Isolation mobiler Programme beschrieben. Sie basiert auf der Angabe von signierten Wunsch-Listen, die dem mobilen Programm hinzugefügt und zusammen mit diesem auf ein Rechnersystem geladen werden. Diese Listen werden von dem System als Grundlage für die Überprüfung aller Zugriffe auf Systemressourcen benutzt. In den Listen werden symbolische Namen verwendet; die Einträge sind also unabhängig von dem jeweils verwendeten Rechnersystem.

Die Technik wurde an einer Implementation in ein UNIX-System demonstriert. In weiteren Arbeiten soll diese Technik auch durch Implementierungen an anderen Stellen von Systemarchitekturen demonstriert werden.

Literatur

[1] D.S. Milojicic, M.Condict, F. Reynolds, D. Bolinger and P. Dale. Mobile Objects and Agents. *Advanced Topics Workshop at 2nd USENIX COOTS*, 1996

[2] David Tennenhouse. Active Networks. *Second Symposion on Operating System Design and Implementation*, Oct. 1996

[3] Hermann Härtig and Oliver C. Kowalski and Winfried E. Kühnhauser. The BirliX Security Architecture. Journal of Computer Security, Vol. 2, 1993

[4] D.E. Bell and L. LaPadula. Secure Operating Systems: Unified exposition and Multics interpretation. Technical Report 2997, MITRE, March 1976

[5] J.P. Wack and L.J. Carnahan. Keeping Your Site Comfortably Secure: An Introduction to Internet Firewalls. *NIST Special Publication 800-10* U.S. Department of Commerce, National Institute of Standards and Technology, 1994

[6] W.A. Wulf, R. Levin and C. Pierson. Overview of the Hydra operating system development. *Proceedings of the 5th ACM Symposium on Operating System Principles*, 1975, Operating Systems Review

[7] M.V. Wilkes and R.M. Needham. The Cambridge CAP computer and its Operating System. Computer Science Library, 1979

[8] A.S. Tanenbaum, S. Mullender and R. van Renesse. Using sparse capabilities in a Distributed Operating System. *Proceedings of the 6th International Conference on Distributed Computing Systems*, 1986, IEEE Computer Society Press

[9] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg and J. Wolter. The Performance of Micro-kernel based Systems. Accepted at *16^{th} ACM Symposium On Operating System Principles (SOSP)*, 1997

[10] J. Liedtke. Clans & Chiefs. *12. GI/ITG-Fachtagung Architektur von Rechnersystemen*, Kiel 1992, Springer.

[11] H. Härtig and L. Reuther. Encapsulating Mobile Objects. *Proc. of the 17th International Conference on Distributed Computing Systems* Baltimore 1997, pp. 355-362, IEEE Computer Society Press

[12] I. Goldberg, D. Wagner, R. Thomas and E. Brewer. A Secure Environment for Untrusted Helper Applications. *Proc. of the Sixth USENIX UNIX Secutity Symposium*, 1996