Großer Beleg zum Thema: Wiederverwendung vertrauensunwürdiger L4Linux Anwendungen in einer vertrauenswürdigen Mikrokern-Umgebung in DROPS

Jens Syckor
js712688@os.inf.tu-dresden.de
Technische Universität Dresden
Fakultät Informatik
Lehrstuhl für Betriebssysteme

9. März 2004

Inhaltsverzeichnis

1 Einleitung

Wiederverwendung vorhandener Softwarekomponenten hat nicht nur in der Softwaretechnologie einen hohen Stellenwert. In Verbindung mit den Schwerpunkten Integrität und Vertrauenswürdigkeit erhält es eine weit komplexere Bedeutung.

Oftmals werden bewährte Module bedenkenlos in neue Projekte übernommen, aber auch der Entwickler der Software kann oft nicht alle Anwendungsfälle vorausdenken, in denen sein Programm eingesetzt wird. Gerade deshalb müssen in sicherheitskritischen Anwendungen bei erneuter Nutzung vorhandener Komponenten die Integrität und Vertrauenswürdigkeit genauer betrachtet werden.

In meiner Arbeit untersuchte ich die Wiederverwendung vertrauensunwürdiger Linux-Anwendungen in einer vertrauenswürdigen Mikrokern-Umgebung. Die entstandene Architektur baut auf dem Mikrokern Fiasco auf, der die Spezifikation L4 implementiert und nutzt Software aus der aktuellen L4Env Schicht, sowie das Fenstersystem DOpE [6].

Drei Beispielszenarien wurden entworfen, insbesondere Digitales Rechte Management (DRM) ist in diesem Zusammenhang beleuchtet wurden. Darunter versteht man ein System zum Schutze hochwertiger digitaler Medien, das die Kontrolle der Verteilung und die Nutzung dieser Medien beinhaltet [5].

Als Beispielanwendung wurde der “Presenter” entwickelt, ein L4 Server, der Präsentationen bestehend aus Folien visualisieren kann, ähnlich Microsofts PowerPoint Anwendung.

Aufbau der Arbeit Das folgende Kapitel zeigt den aktuellen Stand der Entwicklung vor Beginn des Beleges und erläutert, welche vorangegangenen Ideen in die aktuelle Arbeit eingeflossen sind. Im Kapitel “Entwurf” wird die entstandene Architektur erläutert, alle Komponenten werden vorgestellt und drei Szenarien diskutiert, in denen das Konzept eingesetzt werden kann. Einige interessante Implementierungsaspekte wurden im darauf folgenden Kapitel ausgewählt und näher beleuchtet. Weitergehend ist in Kapitel 5 eine analysierende Leistungsbewertung zu finden. Der Abschluss der Arbeit bildet eine Zusammenfassung mit Ansätzen für weitere Entwicklungen.

2 Stand der Technik

Das folgende Kapitel soll einen Einblick geben, welche Softwarekomponenten und Ideen zu Beginn meines Beleges vorhanden waren, die mir beim ersten Entwurf bzw. der ersten Analyse geholfen haben.

2.1 File-Provider fprov-l4

Um die Funktionalität anzubieten, dass ein beliebiger L4 Server Daten von einer L4Linux-Instanz zur Laufzeit laden kann, gab es bereits vor meinem Beleg mehrere Ansätze. Eine Implementierung ist der im L4 Loader enthaltene fprov-l4 [1]. Der L4 Loader wird benutzt, um zur Laufzeit andere L4 Server zu laden, zu starten und auch wieder zu beenden. Dabei stützt er sich u.a. auf den File-Provider fprov-l4 ab.

Fprov-l4 ist ein L4Linux-Programm, das durch eingeblendete Bibliotheken bestimmte Fähigkeiten eines L4 Servers hat. So kann es z.B. mit dem Name-Server kommunizieren und sich dort anmelden, außerdem kann es Dataspaces erzeugen und an andere L4 Server übertragen. Es basiert auf der Implementierung einer generischen Schnittstelle, die die Funktion open anbietet. Mit open kann ein L4 Server den fprov-l4 ansprechen, indem er u.a. den Namen der zu ladenden Datei übergibt.

Mit dem fprov-l4 ergab sich es eine gute Möglichkeit, das Einladen von Daten zu testen und einige wichtige Teile in die Implementierung der read-Funktion des neu entwickelten File-Providers einfließen zu lassen.

2.2 DOpE-Presenter

Für das Fenstersystem DOpE wurde als Beispielapplikation ein einfacher Presenter entwickelt. Er konnte Grafiken im BMP-Bildformat anzeigen, die alle eine einheitliche Größe hatten, d.h. eine Skalierung war nicht vorgesehen.

Weiterhin konnte nur eine Präsentation angezeigt werden, da die Dateien direkt in der Konfiguration des Bootloaders Grub eingetragen wurden. Bei einem Wechsel zu einem anderen Vortrag mußte deshalb der Rechner neu hochgefahren werden. Die Folien wurden vom Bootloader als Module zur Verfügung gestellt. Außerdem konnte die einmal gestartete Applikation die Reihenfolge der geladenen Folien nicht mehr verändern. Es fand keine Betrachtung im Kontext der Vertraulichkeit statt, der DOpE-Presenter hatte die Intention, möglichst einfach eine Präsentation einzuladen und anzuzeigen.

Dennoch konnte für den neu entworfenen Presenter in der Anfangsphase die Interpretation und Konvertierung des BMP-Bildformates übernommen werden, um ein rasches Prototyping zu ermöglichen.

3 Entwurf

Dieses Kapitel erläutert die entwickelte Architektur zur Umsetzung der Aufgabenstellung. Anschließend werden drei grundlegende Szenarien näher betrachtet.

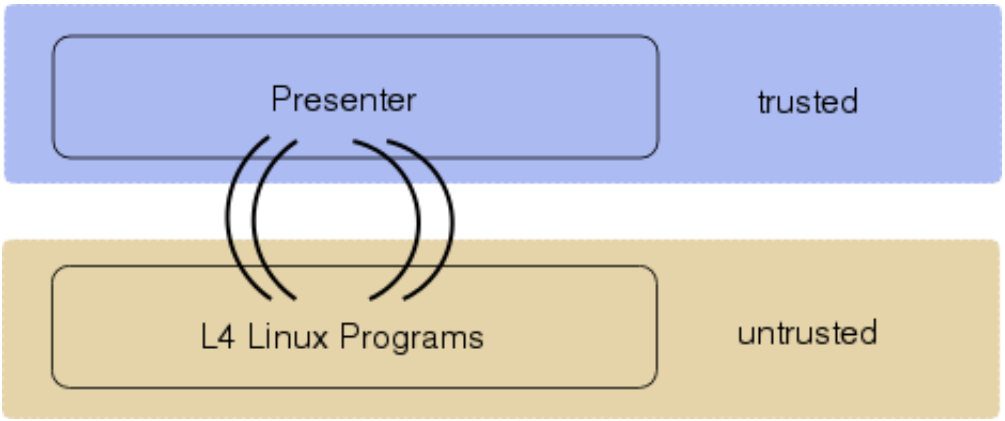

Abbildung 1: Einordnung von Komponenten in vertrauenswürdige bzw. vertrauensunwürdige bzgl. des Tunneling

Um eine vertrauenswürdige Umgebung zu erzeugen, bedarf es einer Verankerung in der Hardware. Da Teile des Systems, die sicheres Urladen (secure booting) und eine TCPA-Infrastruktur [3] ermöglichen, noch nicht implementiert sind, müssen sie als abstrakt vorhanden definiert werden. Im weiteren Verlauf sollen deshalb diese Aspekte keiner weiteren Betrachtung unterliegen.

Die erzeugte Architektur ist eine Umsetzung der Idee des Tunneling aus [7]. Als Tunneling wird eine Technik beschrieben, vorhandene bewährte Software zu benutzen und mit einer Eigenschaft zu erweitern, die von ihr nicht angeboten wird. Die nicht vorhandene Eigenschaft wird von einer zusätzlichen Software erbracht, die nach außen damit die vorhandene Software tunnelt.

Ein Beispiel für die Umsetzung der Tunnelidee im Kontext der Integrität und Vertraulichkeit ist der Aufbau eines Tunnels mit IPsec über unsichere Netzwerke, die standardmäßig nur IP anbieten. IP ist ein Protokoll, das sich nicht um die Datensicherheit im kryptographischen Sinne kümmert. Sie wird von IPsec erbracht, in dem es auf IP gesetzt wird.

Eine erste grobe Annäherung an das System erzeugt eine Einteilung in vertrauenswürdige und vertrauensunwürdige Komponenten. Eine Komponente wird dann als vertrauenswürdig betrachtet, wenn sie nur die erwartete Funktionalität erbringt und nichts zusätzlich, das bedeutet keine versteckten Methoden enthält und nur in der geforderten Folge von Interaktionen mit anderen Teilen der Architektur zusammenarbeitet. Obwohl es schwer zu Implementierungszeit überprüft werden kann, sollten alle Möglichkeiten bedacht werden, um eventuell entstehende verdeckten Kanäle in vertrauenswürdigen Komponenten auszuschließen.

Unter dem Gesichtspunkt der Vertraulichkeit sollen in den folgenden Unterkapiteln die zur Architektur gehörenden Komponenten näher vorstellt werden.

3.1 Legacy L4Linux

Um eine Verbindung zur Außenwelt zu haben, d.h. einen Zugang zum Netz, aber auch einen lokalen Zugriff auf die Geräte der verwendeten Maschine wie z.B. die Festplatte, wird in der entwickelten Umgebung eine L4Linux-Instanz gestartet, die genau solche und weitere Fähigkeiten anbietet. Als komplexes Softwaresystem muss es durch die Offenheit zur Umwelt als vertrauensunwürdig betrachtet werden. Es dient zur Speicherung öffentlicher, z.B. verschlüsselter oder signierter Daten.

Für den Transfer der Daten in den vertrauenswürdigen Teil der Architektur ist ein Provider nötig, der sowohl die Fähigkeiten einer normalen Linux Anwendung hat, als auch von Servern aus der L4-Welt ansprechbar ist. Es wurde ein File-Provider basierend auf [1] entwickelt, der auf einem open/read/close-Schema basiert und mit Dataspaces [4] arbeitet.

Da die vertrauenswürdigen Komponenten mit dem Legacy L4Linux nur ein Portal zur Umgebung haben, lassen sich Denial of Service Attacken nicht ausschliessen.

Das Legacy L4Linux wird in Bezug auf das Tunneling in den vertrauensunwürdigen Teil der Architektur eingeordnet.

3.2 Encapsulated L4Linux

Um vertrauensunwürdige Standard-Linux-Anwendungen in einem vertrauenswürdigen Teil des Systems wiederzuverwenden, wurde das Encapsulated L4Linux entwickelt.

Obwohl es sich in einer vertrauenswürdigen Umgebung befindet, wird es von allen anderen L4 Servern als klar vertrauensunwürdig eingestuft. Eine solche Einstufung muss getroffen werden, weil sich im Encapsulated L4Linux beliebige Linux Anwendungen befinden, die nicht auf Vertrauenswürdigkeit verifiziert worden sind. Sie sollen bewusst ohne explizite Quellenüberprüfung benutzt werden können.

Da dem Encapsulated L4Linux nicht vertraut werden kann, wurde entschieden, es in einer RAM-Disk arbeiten zu lassen. Um die Verbindungen zur Außenwelt noch weiter einzuschränken, hat die Komponente keinen Zugriff auf physische Hardware, alle Gerätetreiber wurden deaktiviert. So besteht z.B. kein Zugriff auf die Tastatur oder die Netzwerkkarte.

Die Komponente ist somit ein minimales L4Linux, das mit den Tools gebaut wird, die gerade für die Arbeit auf der vertrauenswürdigen Seite gebraucht werden, d.h. wiederverwendet werden sollen.

Denkbar wäre z.B. ein komplexer Codec für ein Video, der nur im vertrauenswürdigen Teil der Architektur benutzt werden darf, da die Schlüssel für das Video nur hier vorhanden sind. Weil der Codec nicht neu programmiert werden soll, wird er in die aktuelle RAM-Disk eingebaut. Im meiner Implementierung enthält das Encapsulated L4Linux die Anwendungen convert und ghostscript, um aus Postscript-Dateien PNG-Bilder zu erzeugen. Im Kontext der Idee des Tunneling

gehört das Encapsulated L4Linux in Abbildung 1 zum vertrauensunwürdigen Teil der Architektur, die Eigenschaft der Vertrauenswürdigkeit wird einer Komponente erbracht, die sich in der vertrauenswürdigen Schicht befindet. Eine Grundvoraussetzung dafür ist der Einsatz einer Mikrokernarchitektur, die Methoden wie separate Adressräume zur Verfügung stellt. Weiterhin wird das Encapsulated L4Linux vollkommen vom Presenter kontrolliert, alle Ein- und Ausgaben sind vom ihm abhängig.

Um die weiterverwendeten Applikationen in der RAM-Disk ausführen zu können, wird ein Programm benötigt, das ähnlich dem File-Provider gleichzeitig L4Linux Fähigkeiten besitzt und ein L4 Server ist. Ich habe mich entschieden, eine execv-Funktion in einem L4Linux-Programm zu implementieren, um diese Funktion anderen L4 Server nach außen über eine Schnittstelle zugänglich zu machen. Mit einem execv-Aufruf soll ein möglichst einfacher, bekannter und flexibler Mechanismus für den Zugriff auf die Anwendungen bereitgestellt werden.

3.3 Presenter

Der Presenter ist ein Darstellungsprogramm für die Folien einer Präsentation. Er ist der L4 Server, der den Mittelpunkt der vertrauenswürdigen Seite darstellt. Er wird benötigt, um die ihm zur Verfügung gestellten Daten zu interpretieren und hat als einzige Instanz die Erlaubnis, daraus gewonnene Informationen darzustellen. Er allein ist es auch, der die Verbindungen zum Legacy L4Linux sowie zum Encapsulated L4Linux verwaltet und benutzt.

Als weitere Aufgabe obliegt dem Presenter die Kommunikation mit einem Schlüsselserver, der alle notwenigen Schlüssel verwaltet und persistent ablegt.

Der Presenter benötigt komplexe Funktionen, um z.B. das Konvertieren der Folien in das von ihm benutze Format PNG durchzuführen. Für solche und weitere Aufgaben gibt es bereits etablierte Softwarelösungen für Linux. Um den Implementierungsaufwand zu minimieren und die vertrauenswürdigen Teile im Presenter so klein und überschaubar wie möglich zu halten, wurden komplexe Funktionen ausgelagert. Der Presenter nutzt das Encapsulated L4Linux, um dort Programme, die solche Teilaufgaben übernehmen und damit wiederverwendet werden, über die oben erwähnte execv-Schnittstelle aufzurufen.

Die Vertraulichkeit bezüglich des Encapsulated L4Linux wird für den Presenter deshalb möglich, weil diese L4Linux Instanz keine Verbindung zur Hardware hat und von außen daher keine Manipulationen möglich sind.

Zur Darstellung der Dokumente, d.h. in meinem Fall der Folien im PNG Bildformat, wird eine vertrauenswürdige grafische Oberfläche vorausgesetzt [7]. Der Presenter verwendet dafür das Fenstersystem DOpE [6], das durch seine geringe Code-Komplexität von weniger als 10000 Zeilen als vertrauenswürdig angesehen wird.

Im Konzept des Tunneling stellt der Presenter in Abbildung 1 die Komponente dar, die die zusätzliche Funktionalität der Vertrauenswürdigkeit erbringt.

3.4 Sicherer Speicher

Für die Persistenz sicherheitsrelevanter Informationen wird eine zusätzliche vertrauenswürdige Komponente benötigt, die eine solche Schnittstelle implementiert. Sie sollte weiterhin eine Schlüsselmanagement Infrastruktur anbieten, um schnell und flexibel auf die privaten Schlüssel und weitere vertrauliche Daten zugreifen zu können.

Der Sichere Speicher, im weiteren secure storage, ist eine Hilfskomponente zur Erbringung der zusätzlichen Eigenschaft der Vertrauenswürdigkeit im Kontext des Tunneling.

3.5 Szenarien

Das entstandene Konzept kann in einer Vielzahl von Szenarien eingesetzt werden, von denen in dieser Arbeit drei näher untersucht wurden. Das Hauptkriterium für die Auswahl der folgenden Szenarien war die Betrachtung unter dem Gesichtspunkt Vertraulichkeit, speziell DRM und Integrität in Bezug auf Signierung und deren Wechselwirkungen mit der entwickelten Architektur. Jedes Szenario sollte auf mögliche Angriffsformen untersucht werden. Die verwendete Notation ist an [8] angelehnt, es werden folgende Symbole eingeführt:

x unverschlüsselte Rohdaten

Sig Signatur

x,Sig mit der Signatur Sig signierte Daten x

sign Signaturfunktion

test Testfunktion für die Überprüfung der signierten Daten

s privater Schlüssel

t öffentlicher Schlüssel

c(x) verschlüsselte Daten

Signieren von Daten Im ersten Szenario soll die Architektur benutzt werden, um Daten, die aus dem vertrauensunwürdigen Legacy L4Linux geladen wurden, auf der vertrauenswürdigen Seite mit Hilfe des Encapsulated L4Linux zu signieren. Damit kann Alice die Integrität der Daten x erzeugen und Bob einen Nachweis liefern, dass x nur von Ihr stammen kann. Falls x auf dem Weg zu Bob manipuliert wurde, hat er die Möglichkeit, die Veränderung festzustellen. Bei dem

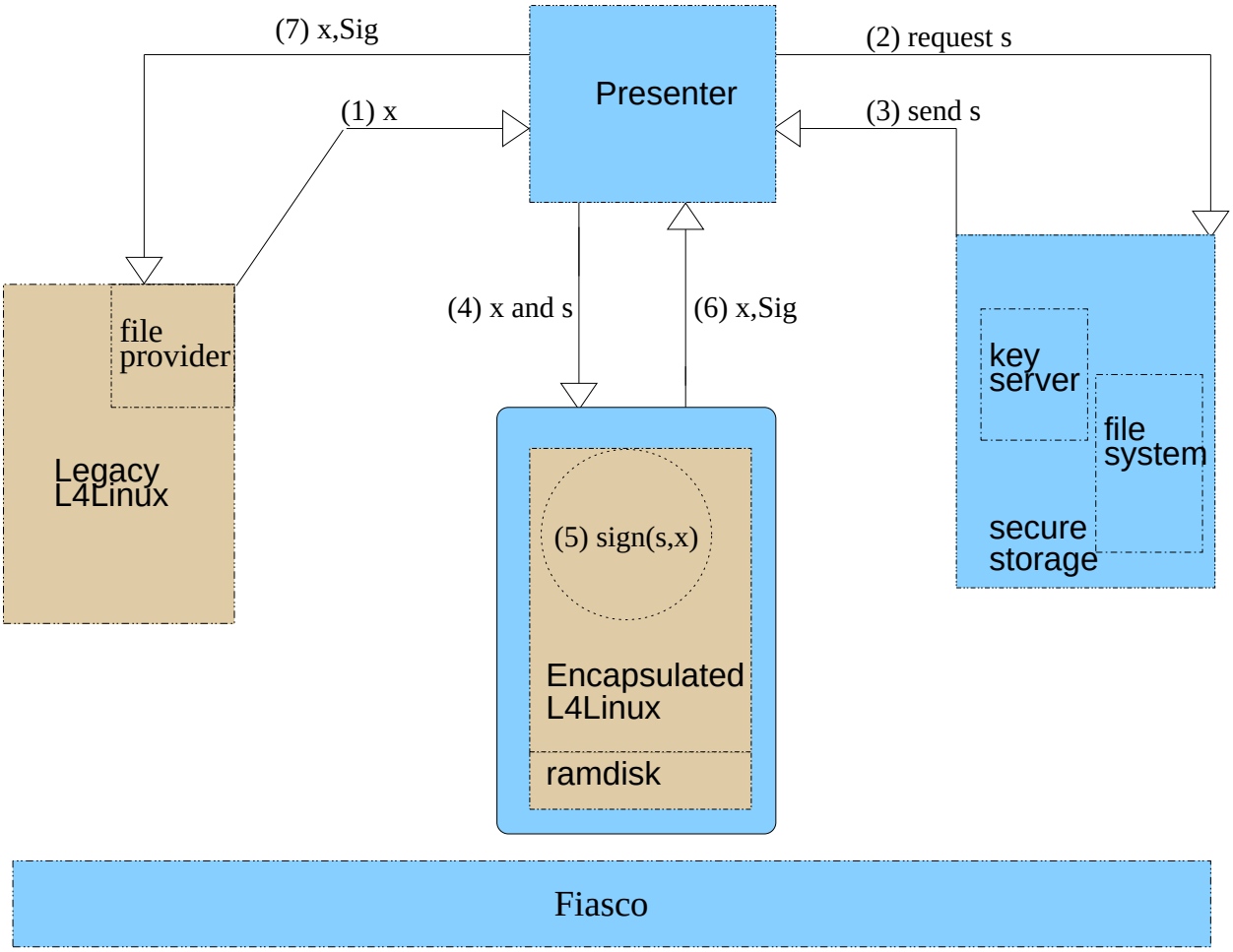

Abbildung 2: Interaktionen und Architektur des Szenarios “Signieren von Daten”

betrachteten Verfahren soll es sich um ein digitales Signatursystem handeln. Bob hat den öffentlichen Schlüssel t von Alice, um x,Sig (Sig ist die Signatur) mit test(t,x,Sig) zu überprüfen. Bob erhält mit der Funktion test x, die Signatur Sig und das Ergebnis, ob die Daten wirklich von Alice signiert, d.h. nicht verändert wurden.

Die Idee, die das Szenario so interessant macht, ist die Wiederverwendung einer vertrauensunwürdigen Signierfunktion im Encapsulated L4Linux in einer Architektur, die gerade vertrauenswürdige Aktionen erzeugen will.

Im obigen Beispiel bedeutet es, dass Alice x,Sig durch sign(x,s) erzeugt hat, wobei s der private Schlüssel von Alice und sign die wiederverwendete vertrauensunwürdige Signierfunktion ist, die sich im Encapsulated L4Linux befindet.

Anhand von Abbildung 2 möchte ich den Datenfluss erläutern:

Um x zum Presenter zu transferieren, wird der unter dem Legacy L4Linux gestartete File-Provider beim Name Server gesucht. Falls er angemeldet ist, wird er mit einem open-Aufruf vom

Presenter angesprochen. Der File-Provider gibt ein lokales Handle auf x an den Presenter zurück. Er kann dann mit Hilfe des Handles eindeutig auf x zugreifen und sich x durch einen read-Aufruf vom File-Provider holen (1).

Im Schritt (2) fordert der Presenter den privaten Schlüssel s zum Signieren von der Schlüsselmanagement-Infrastruktur an, die zur Komponente secure storage gehört. Nachdem er s in (3) erhalten hat, schickt er in (4) x und s an das Encapsulated L4Linux.

Dort wird über die execv-Schnittstelle die vertrauensunwürdige Signierfunktion sign ausgeführt, d.h. durch sign(x,s) wird x,Sig erzeugt (5). Daraufhin wird x,Sig an den Presenter zurückgeschickt (6). Im letzten Schritt speichert der Presenter x,Sig im vertrauensunwürdigen Legacy L4Linux ab. Er ruft dazu den File-Provider durch open/write auf und übergibt ihm x,Sig. Jetzt ist x,Sig im Legacy L4Linux und kann danach von Alice an Bob gesendet werden.

Da die wiederverwendete vertrauensunwürdige Signierfunktion sign mit Hilfe des Encapsulated L4Linux ausgeführt wird, muss folgende Randbedingung beachtet werden:

Jede Instanz des Legacy L4Linux kann nur für die exklusive Anwendung eines geheimen Schlüssels benutzt werden. Sobald zwei Schlüssel in die Komponente übertragen worden sind, ergibt sich die Angriffsmöglichkeit, eine gültige Unterschrift unter ein falsches Dokument zu setzen.

Außerdem sollte beachtet werden, dass das Encapsulated L4Linux eine DoS Attacke durchführen kann, indem es eine beliebige, zufällige Unterschrift unter die Daten x setzt. Eine weiterer Angriff ist die Funktion sign so zu manipulieren, dass sie ein zufälliges x erzeugt und mit s unterschreibt.

Weitere Untersuchungen führten zu dem Ergebnis, dass es nicht sinnvoll ist, den privaten Schlüssel s an das vertrauensunwürdige Encapsulated L4Linux zu übergeben. Ein Angreifer hat sonst die Möglichkeit, die Funktion sign so zu manipulieren, dass s in die signierte Nachricht x,Sig eingebettet wird. Zur Abwehr des Angriffes wird die vertrauliche Testfunktion test benötigt, die x,Sig dahingehend überprüft, ob s in x,Sig eingebettet wurde. Da test im allgemeinen Fall die gleiche Komplexität wie sign besitzt, gewinnt man durch die Wiederverwendung der vertrauensunwürdigen Funktion sign nichts hinzu, d.h. man wird gleich sign in einer vertraulichen Komponente implementieren wollen. Nur für den Fall, dass test eine deutlich geringere Komplexität als sign besitzt, d.h. sehr schnell und einfach durchzuführen ist, wird das Szenario eine praktische Relevanz haben.

Überprüfung der Integrität von Daten Im zweiten Szenario soll untersucht werden, ob die Integrität von x auf vertrauenswürdige Weise durch den Einsatz des entwickelten Konzeptes nachgewiesen werden kann. Erneut wird ein digitales Signatursystem als Grundlage verwendet.

Bob möchte mit Hilfe der Architektur überprüfen, ob das im ersten Szenario erzeugte x,Sig auch wirklich von Alice stammt und auf dem Weg zu ihm nicht manipuliert wurde. Er erhält durch den

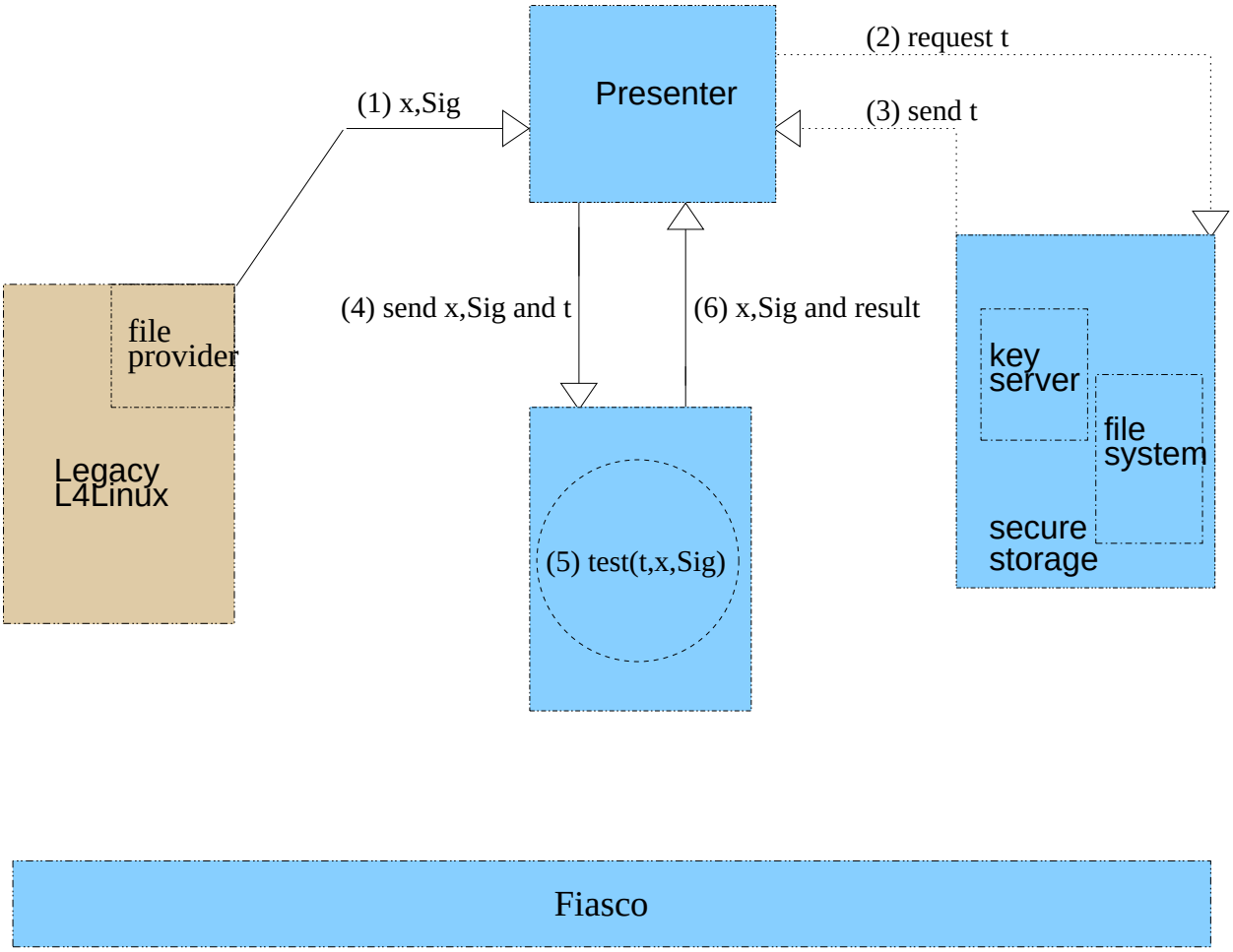

Abbildung 3: Interaktionen und Architektur des Szenarios “Überprüfung der Integrität von Daten”

Einsatz der Komponenten als Ergebnis x, die Signatur Sig und erfährt, ob x von Alice signiert und unterwegs nicht manipuliert wurde.

Abbildung 3 zeigt die Interaktionen der einzelnen Teile und soll nachfolgend erläutert werden.

In Schritt (1) holt sich der Presenter mit den im ersten Szenario erläuterten Teiloperationen via eines open/read-Aufrufes die Daten x,Sig vom vertrauensunwürdigen Legacy L4Linux. Im Schritt (2) und (3) fordert der Presenter den notwendigen Schlüssel t von der Komponente secure storage an. Diese beiden Schritte sind streng genommen nur in einem symmetrischen Authentikations-system notwendig, weil die Schlüssel beider Partner in einem solchen Verfahren geheimgehalten werden müssen. Die Aktionen wurden deshalb eingefügt, weil die Konvention getroffen wurde, alle Schlüssel in der Komponente secure storage persistent zu halten und der Presenter bei der Schlüsselsuche immer dort nachschaut. Bei der Betrachtung eines digitalen Signatursystems kann der öffentliche Schlüssel t auch vom Legacy L4Linux geholt werden.

Es stellt sich erneut die Frage, ob bewährte Softwarelösungen, die aber vertrauensunwürdig sind, im Encapsulated L4Linux genutzt werden können, um zu überprüfen, ob x integer ist. Nähere Untersuchungen brachten das Ergebnis, dass der Einsatz vertrauensunwürdiger Software für diesen Zweck nicht möglich ist.

Der Einsatz bewährter Anwendungen würde den fundamentalen Angriff ermöglichen, bei dem die Überprüfung der Integrität von x mit dem jeweils falschen Ergebnis beantwortet wird. Bob könnte also mit einer Wahrscheinlichkeit nicht besser als 0.5 sagen, ob x von Alice unterschrieben und von einer dritten Partei nicht manipuliert wurde. Das Beispiel soll verdeutlichen, das wenn wiederverwendete Programme nicht die Eigenschaft haben, Integrität eindeutig festzustellen, auch durch die zusätzliche Komponente Presenter diese Funktionalität nicht erbracht werden kann. Tunneling bringt also keine Verbesserung.

Deshalb wird in (4) x,Sig und t an eine zusätzliche vertrauenswürdige Komponente geschickt, die durch die Funktion test(x,Sig,t) eindeutig feststellen kann, ob x nicht manipuliert wurde und Alice es signiert hat. Die zusätzliche Komponente sendet in (6) x, die Signatur Sig und das Ergebnis ob, x integer ist, an den Presenter. Danach kann Bob entscheiden, ob er x vertraut oder nicht.

DRM Szenario Im dritten Anwendungsfall will Alice die Architektur einsetzen, um die Daten x, für die sie bei einem Content Provider bezahlt hat, im Legacy L4Linux abzulegen. Danach will sie x auf der vertrauenswürdigen Seite mit Hilfe eines Schlüssels d entschlüsseln und das Encapsulated L4Linux benutzen, um auf x Aktionen auszuführen.

Es ist zu untersuchen, ob das Konzept in einem DRM orientierten Kontext eingesetzt werden kann, d.h. ob es insbesondere möglich ist, bewährte Softwarelösungen zum Ausführen von Aktionen auf den entschlüsselten Daten wiederzuwenden, ohne die Definition der Vertraulichkeit zu verletzen.

Während in den beiden ersten Szenarien nur die Vertraulichkeit für den Anwender nachgewiesen werden mußte, kommt im Kontext DRM der Content Provider hinzu. Er wird die Daten nur dann an Alice schicken, wenn er der gesamten Architektur vertraut. Diese Vertraulichkeit muss nachgewiesen werden.

Eine konkreter Fall für die Anwendung wäre eine verschlüsselte MP3-Datei, für die Alice bezahlt hat. Das Encapsulated L4Linux soll zur Extrahierung des MP3-Stromes eingesetzt werden, damit sich Alice das Musikstück im Presenter anhören kann.

Mit Hilfe der Abbildung 4 sollen die entstehenden Interaktionen näher erläutert werden:

In Schritt (1) liefert der Content Provider, der zur Vereinfachung als außerhalb des Systems stehend betrachtet wird, die verschlüsselten Daten c(x) an das Legacy L4Linux. Danach lädt sich der Presenter in (2) c(x) vom Legacy L4Linux mit dem im ersten Szenario geschilderten Verfahren. Für die Entschlüsselung von c(x) benötigt der Presenter den Schlüssel d, den er sich in (3) und (4) vom secure storage holt.

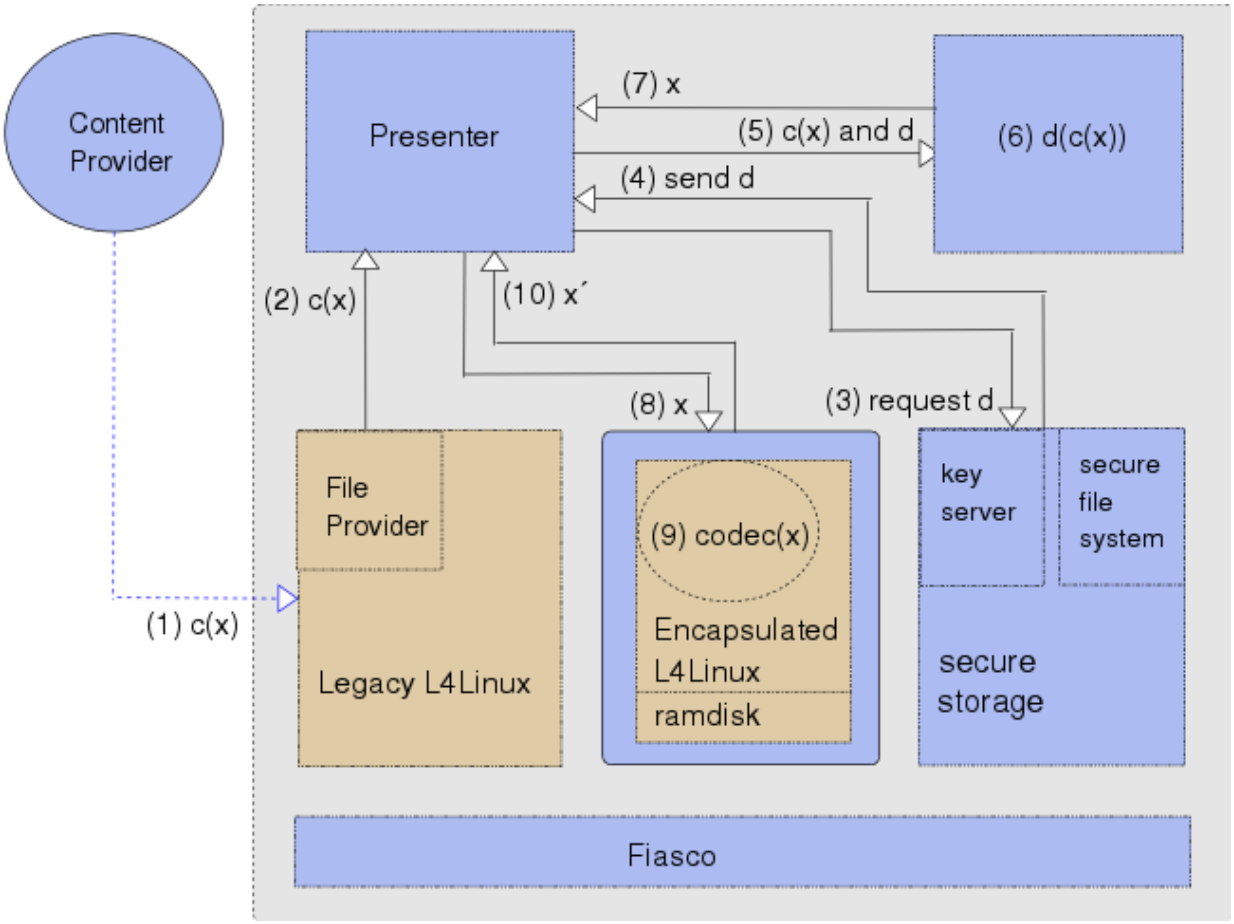

Abbildung 4: Interaktionen und Architektur des DRM Szenarios

Danach übergibt er c(x) und d an einen vertraulichen Kryptographie-Server, der mit d(c(x)) die Daten x entschlüsselt und x an den Presenter zurückliefert.

Nun überträgt der Presenter x an das Encapsulated L4Linux und startet in (9) eine wiederverwendete Applikation, die eine Aktion auf den Rohdaten x ausführt. Dann erhält er die verarbeiteten Daten x' zurück.

Damit der Presenter im Sinne des Tunneling die Eigenschaft der Vertrauenswürdigkeit für die bearbeiteten Daten x' erbringen kann, benötigt er weitere Informationen, um zu überprüfen, dass x korrekt verarbeitet wurde.

Sind die Informationen nicht vorhanden, kann das Encapsulated L4Linux einen Angriff ausführen, indem es anstatt x' ein x" zurückgibt, das vertrauliche Informationen enthält. In konkreten Fall könnte die wiederverwendete Applikation z.B. anstatt eines Videos einen Bitstream zurückgeben, der das Video repräsentiert. Der Presenter hat kein Wissen zu überprüfen, ob x'=x" ist und würde x" ohne Bedenken darstellen.

Eine Möglichkeit, um den Angriff zu verhindern, ist ein Hashcode über x'. Der Presenter hat mit dem Hashcode die Möglichkeit zu nachzuweisen, ob x'=x". Notwendig für diese Sicherheitsüberprüfung ist ein Hashcode, der vom Content Provider geliefert werden muss, denn nur er hat das initiale Wissen, wie x' auszusehen hat. Der Hashcode gelangt via dem Legacy L4Linux zum Presenter, der im letzten Punkt h(x') bildet und mit dem erhalten Hashcode vergleicht. Nur wenn die Software im Encapsulated L4Linux korrekt gearbeitet hat, stimmt x' mit x" überein. Der Presenter kann in diesem Fall x' vertrauen und es weiterverarbeiten.

Aus dem vorgeschlagenen, "einfachsten" Weg der Lieferung des Hashcodes an den Presenter ergibt sich direkt eine neuer Angriff. Da sowohl das Legacy L4Linux als auch das Encapsulated L4Linux als vertrauensunwürdig definiert wurden, besteht bei einer Zusammenarbeit beider Komponenten die Möglichkeit, den Hashcode, der an den Presenter geliefert wird, so zu manipulieren, dass er h(x") entspricht. Damit wird dem Presenter glaubhaft gemacht, das x'=x".

Es muss also garantiert sein, dass die Komponenten Legacy L4Linux und Encapsulated L4Linux nicht miteinander zusammenarbeiten können. Ein Grundvoraussetzung für die Sicherheit bietet die Forderung, dass der File-Provider keine verdeckten Kanäle benutzt und nur die Aufgaben erledigt, die vom ihm verlangt werden.

Ein Lösungsansatz für die Vermeidung solcher Angriffe bietet die Möglichkeit, den Hashcode verschlüsselt vom Content Provider an den Presenter zu schicken.

4 Implementierung

Im folgenden Kapitel werden einige interessante Aspekte der Implementierung herausgegriffen und näher erläutert. Es sollen die wichtigsten Grundgedanken für den Entwurf des Presenters dar-

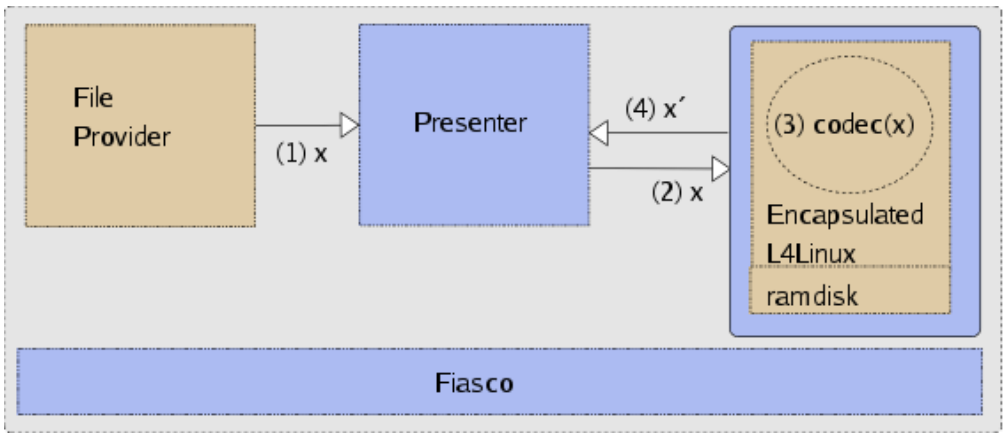

Abbildung 5: Implementiertes Postscript-Szenario

gelegt, der File-Provider und das execv-L4-Programm diskutiert werden. Weiterhin möchte ich die Vorgehensweise bei der Portierung der libpng darstellen.

Bei der Implementierung wurde ein Postscript-Szenario umgesetzt, siehe Abbildung 5. Der Presenter kann Daten im Postscript-Format einlesen. Zur Minimierung des vertrauenswürdigen Codes nutzt er zur Konvertierung der Postscript-Daten in einzelne PNG-Bilder die Anwendungen convert, ghostscript und psselect im Encapsulated L4Linux. Mit der Umsetzung sollte gezeigt werden, dass es möglich ist, vertrauensunwürdige bewährte Softwarelösungen wiederzuverwenden, d.h. das die Vertraulichkeit für den Nutzer dadurch gewahrt bleibt.

4.1 L4Linux File Provider

Um Daten an die vertrauenswürdige Seite des Systems zu übertragen, die von der Außenwelt in das vertrauensunwürdige Legacy L4Linux gelangt sind, bzw. um Daten darin abzulegen, bedarf es eines Programms, das eine solche Schnittstelle implementiert und diese Funktionen anderen L4 Servern anbietet. Der entstehende L4 Server muss gleichzeitig ein Linux-Prozess sein, damit er als Schnittstelle zwischen L4Linux und der L4 Welt arbeiten kann.

Der für diesen Zweck konstruierte File-Provider kann gleichzeitig auf L4Linux Bibliotheken zugreifen und ist durch eine eingeblendete Emulationsbibliothek fähig, mit anderen L4 Server zu kommunizieren So kann er sich z.B. beim Name-Server anmelden.

Um die bewährte Unix-Philosophie “Alles ist ein File” für den File-Provider nutzen zu können, implementiert er eine Schnittstelle, die Operationen wie open, read, write und close anbietet.

Abbildung 6 zeigt die Interaktionen zwischen File-Provider und Presenter.

Nachdem der Presenter vom Ressourcen-Manager gestartet wurde, wartet er in Schritt (1) solange, bis der File-Provider geladen wurde und sich in (2) beim Name-Server angemeldet hat. Nun kann der Presenter mit den File-Provider kommunizieren. Er ruft ihn mit der Funktion open auf,

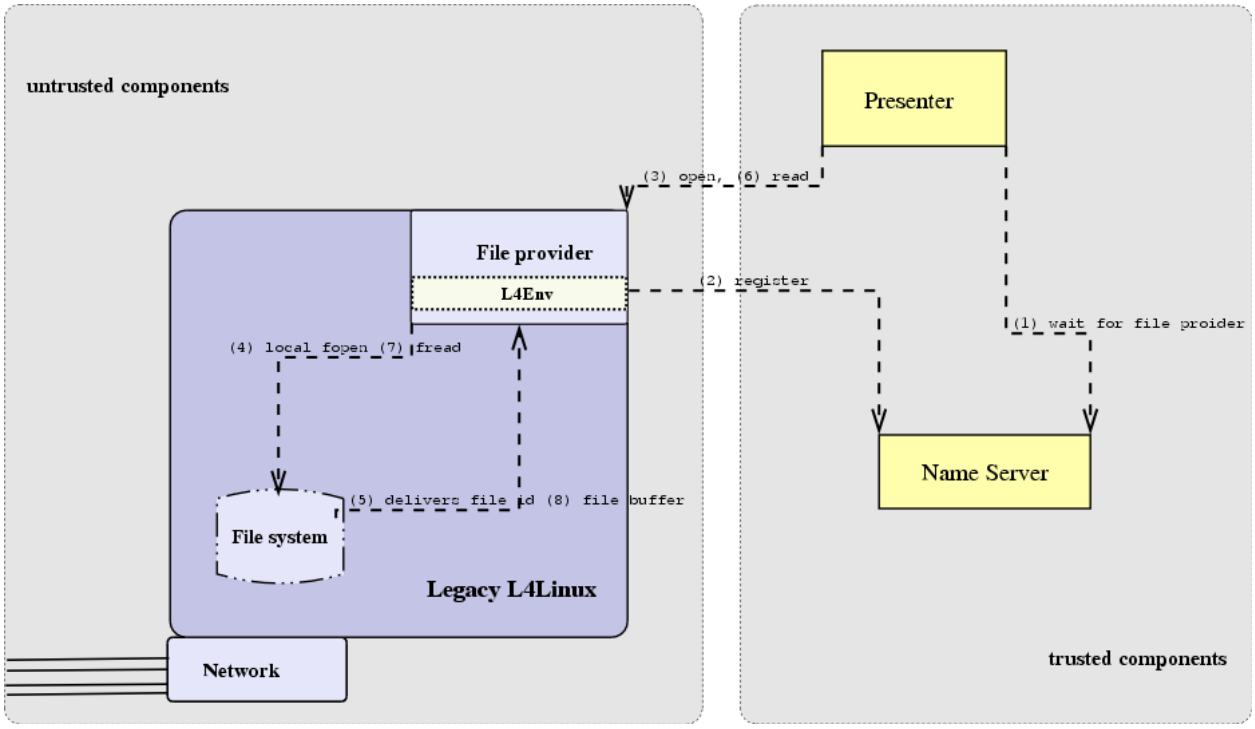

Abbildung 6: Interaktionen zwischen dem Presenter und File-Provider

indem er den Namen der zu ladendenden Datei übergibt. Der File-Provider bildet in (4) das open der IDL-Schnittstelle auf das lokal implementierte open von L4Linux ab und gibt das erzeugte File-Handle an den Presenter weiter. Dieser kann mit dem Handle den Provider erneut aufrufen, nun mit read. Die Operation wird erneut lokal abgebildet, sie lädt die Datei in einen Puffer und übergibt ihn an den Presenter.

Eine Trennung von open und read wurde deshalb vorgenommen, um möglichst flexibel auf die Daten im Legacy L4Linux zuzugreifen, d.h. z.B. mit der Funktion seek kann problemlos mit Hilfe des Handles die aktuelle Leseposition verändert werden.

Eine weitere Designentscheidung war die Verwendung von Dataspaces [4] als zu nutzende Speicherimplementierung, da Dataspaces ein Rechtemodell anbieten, das beim Transfer der Daten auf die vertrauenswürdige Seite verwendet wird. Dataspaces sind Speicherobjekte, die ein high-level-Interface anbieten, neben dem erwähnten Rechtemodell können sie z.B. von mehreren Threads über eine share-Operation gemeinsam genutzt werden.

Beim Schritt (8), d.h. bei einem read, werden die Rechte am Dataspace an den aufrufenden L4 Server übertragen. Danach hat die vertrauensunwürdige Seite keinen Zugriff mehr auf die Daten und der vertrauenswürdige L4 Server, hier wieder der Presenter, kann den Dataspace mit Hilfe des Dataspace-Managers in seinen Adressraum einblenden.

Um Wiederverwendung auch mit den eigenen Programmen zu praktizieren, wird der File-Provider sowohl vom Legacy L4Linux gestartet, als auch im Encapsulated L4Linux. Um die beiden gestarteten Instanzen beim Aufruf zu unterscheiden, melden sie sich mit einem unterschiedlichen Namen beim Name-Server an.

Weiterhin muss beachtet werden, dass der File-Provider die Rechte zum Lesen/Schreiben von Dateien bekommt, von dessen Nutzer er gestartet wurde. Im betrachteten Kontext der Datensicherheit empfiehlt es sich daher genau zu überlegen, in welcher Situation der File-Provider von root gestartet werden soll, bzw. ob dies überhaupt sinnvoll ist. Der File-Provider bekommt nicht automatisch root-Rechte, damit es dem Nutzer selbst überlassen werden kann, welche Sicherheitseinstellungen er benötigt, es wird das Prinzip der geringstmöglichen Privilegierung eingesetzt [8].

4.2 Entwurfsmuster im Presenter

Im folgenden Abschnitt sollen einige Entscheidungen erläutert werden, die für die Entwicklung des Presenter L4 Server getroffen wurden.

Der Presenter wurde in 100% ANSI-C geschrieben, um eine möglichst hohe Portabilität für andere Plattformen zu erreichen. Es wurde ein objektorientierter Programmierstil in Anlehnung an die Implementierung von DOpE gewählt, um die einzelnen Komponenten zu entkoppeln und nur über definierte Schnittstellen zugänglich zu machen.

Da der Presenter eine klassische Anwendung mit ereignisorientierter Oberfläche ist, wurde als Entwurfsmuster eine Model-View-Controller-Architektur umgesetzt, d.h. das jede der drei Schichten durch eine beliebige andere Implementierung ersetzt werden kann.

Denkbar wäre z.B. die Daten nicht via des entworfenen open/read-Schemas zu holen, sondern mit Hilfe einer Implementierung, die auf einer SQL-orientierten Datenbankabfrage basiert. Im genannten Beispiel würde daraus folgen, das das Legacy L4Linux durch eine nicht vertrauenswürdige Datenbank ersetzt werden müsste.

Die Abbildung 7 verdeutlicht den Entwurf der Model-Schicht. Als allgemeine Klasse wurde die Presenter-Klasse erzeugt, die Methoden enthält, um den Namen und den Schlüssel zu verwalten. Die Klassen Slide und Presentation erben diese und sind über eine Assoziation verbunden. Alle Instanzen von Presentation sind über den PresManager erreichbar. Jede Folie erhält einen künstlich erzeugten Schlüssel, damit ist es möglich, auch Folien mit dem selben Namen, die mehrfach in einer Präsentation vorkommen, unabhängig zu verwalten.

Der Controller-Klasse, die für die Darstellung der aktuellen Präsentation verantwortlich ist, wird eine Instanz einer Liste übergeben, die alle Folien in ihrer Reihenfolge enthält. Der Controller muss nur noch über die Liste iterieren und kann die Folien schnell darstellen.

Abbildung 8 zeigt ein Bild der Oberfläche und Menüstruktur. Auf der linken Seite wird eine Vorschau der Folien generiert, damit man einen schnellen Überblick über die Präsentation bekommt.

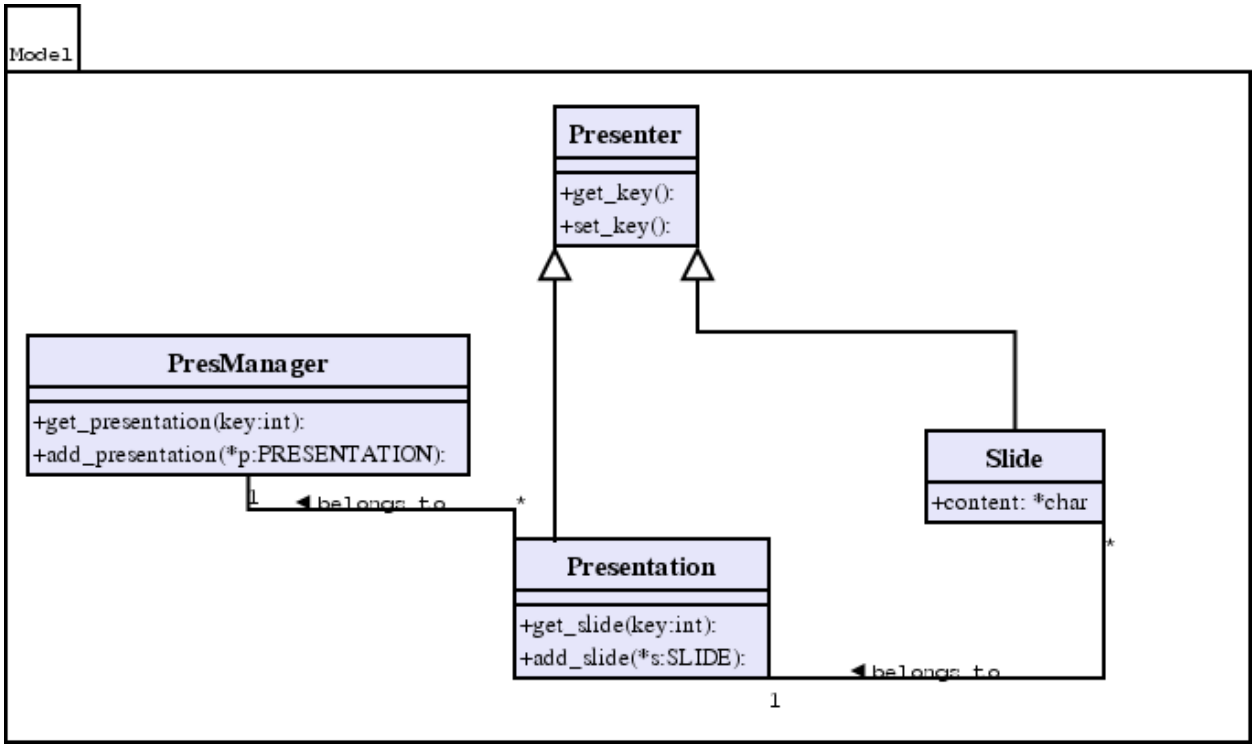

Abbildung 7: Klassen in der Modell-Schicht des Presenters

Abbildung 8: Screenshot der Oberfläche des Presenters

Auf der rechten Seite wird eine verkleinerte Form der aktuellen Folie dargestellt. Um innerhalb des Quelltextes eine hohe Wiederverwendung zu praktizieren und Sonderfälle zu vermeiden, wird für die Generierung der Vorschau und der aktuellen Folie die gleiche Routine benutzt. Folien, die größer als der aktuelle Bildschirmbereich sind, werden skaliert und entsprechend ausgerichtet.

Als Eingangsdaten kann dem Presenter wahlweise eine Konfigurationsdatei übergeben werden, die die Namen aller Folien einer Präsentation enthält, oder ein Pfad auf eine Postscript-Datei.

Im ersten Fall lädt der Presenter alle Folien der in der Konfiguration gegebenen Reihenfolge, die Folien müssen im PNG-Bildformat sein. Der Aufbau der Konfigurationsdatei ist so einfach wie möglich gehalten wurden, es muss lediglich der Pfad auf das die Daten enthaltende Verzeichnis und die Namen der Bilder in der gewünschten Abfolge eingetragen werden. Wiederholungen der selben Folie sind problemlos möglich. Die Konfigurationsdatei ermöglicht es sehr effizient, mehrere Präsentationen zu erstellen, die aus den gleichen Bilder aufgebaut sind, aber eine unterschiedliche Reihenfolge haben oder einfach Bilder in mehreren Präsentationen wiederzuverwenden. Dafür ist lediglich eine neue Konfigurationsdatei notwendig.

Der Presenter kann außerdem Postscript-Datei verarbeiten, indem er sie einliest und in das Encapsulated L4Linux kopiert. Dort befinden sich die wiederverwendeten Linux-Applikationen convert und ghostscript. Der Presenter benutzt sie, um aus dem Postscript einzelne Folien im PNG-Format zu erstellen, aus denen er die Präsentation aufbauen kann.

4.3 Portierung der libpng auf L4

In der Anfangsphase unterstützte der Presenter das Bildformat BMP, weil es durch seinen einfachen Aufbau leicht einzubinden war. Im Verlaufe der Entwicklung stellte sich jedoch heraus, das BMP einige Nachteile hat. So bietet es keine Kompression an und ist sehr ressourcenintensiv. Deshalb habe ich mich entschieden, als Bildformat PNG einzusetzen. Es bietet im Gegensatz zu JPEG verlustfreie Kompression, unterstützt echte Transparenz durch einen Alpha-Kanal, die Möglichkeit in der Datei Informationen wie Autor- oder Copyrighthinweise abzulegen und ist lizenzfrei.

Für das PNG-Format gibt es bereits die frei verfügbare offizielle Referenzbibliothek “libpng” [2]. Sie gestattet es relativ einfach, PNG Dateien einzulesen und vorzuverarbeiten. So ist es z.B. möglich, die Farbpalette zu ändern oder den Alpha-Kanal zu entfernen.

Die libpng Bibliothek liegt im ANSI-C Quellcode vor und wird u.a. in Linux und auf dem Atari eingesetzt. Die Bibliothek benötigt für die Kompression und Dekompression die zlib, die schon für L4 portiert worden war. Auf Grund der gegebenen Vorraussetzungen entschloss ich mich deshalb, die libpng für L4 zu portieren. Um den Quellcode der Bibliothek nicht zu verändern, d.h. sie als Sandbox betrachten zu können, habe ich eine Emulationsumgebung geschrieben, in der einige Funktionen ersetzt werden, auf die die libpng zurückgreift. Es wurde z.B. die Funktion fread so ersetzt, dass sie nun einen Dataspace ausliest, indem sich die PNG-Datei befindet.

Mit ca. 200 Zeilen Code in der Emulationsumgebung konnte die libpng für L4 Server verfügbar gemacht werden.

4.4 Programmausführung unter Encapsulated L4Linux

Im Encapsulated L4Linux wird ein L4 Server als Linux-Prozess mit einer Schnittstelle benötigt, mit dem es möglich wird, unkompliziert auf die weiterzuverwendenden Anwendungen in der RAM-Disk zuzugreifen und diese auszuführen.

Es wurde die Entscheidung getroffen, die execv Funktion von Linux für L4 Server nach außen anzubieten, d.h. ein Programm zu schreiben, das diese Funktion ausführt. Mit dem PRESENTER_EXEC kann man nun direkt eine Software im Encapsulated L4Linux ausführen. Zu verarbeitende Daten müssen vorher per File-Provider in die RAM-Disk übertragen werden. Die Trennung der von execv und read/write ergab sich, da es Probleme gab, execv und Pipes via Dataspaces zu verbinden. Einige Linux Anwendungen brauchen erst die kompletten Daten, bevor sie eine Aktion ausführen, andere wie z.B. cat werten immer erst Daten in der Größe einer Pipe aus und warten dann auf den nächsten Eingabestrom.

Die Trennung hat den Vorteil, dass das Ausführen von Programmen via execv sowie Lesen aus bzw. Schreiben in die RAM-Disk sehr flexibel miteinander verknüpft werden können.

Dem Encapsulated L4Linux wurden wegen der Vertrauensunwürdigkeit alle Zugriffe auf sämtliche Geräte entzogen. Damit der Presenter aber eine Möglichkeit hat, auf eventuelle Fehler- oder Diagnosemeldungen der via execv ausgeführten Programme zu reagieren, werden alle Ausgaben in eine temporäre Datei umgeleitet, auf die der Presenter via open/read mit dem File-Provider zugreifen kann.

5 Leistungsbewertung

Das folgende Kapitel beschäftigt sich mit der Vertrauenswürdigkeit des Presenters bezüglich des Programmumfanges und der Performance beim Darstellen der Präsentationen. Als weiterer Punkt soll der Speicherverbrauch bei ansteigender Zahl der Folien untersucht werden.

Als Testrechner diente ein Intel-Celeron 900Mhz mit 256 MB RAM und einer Matrox MGA G400 AGP Grafikkarte.

5.1 Bewertung der Vertrauenswürdigkeit

Um die Vertrauenswürdigkeit des Presenters einzuschätzen, soll als erstes der Umfang des Quellcodes bewertet werden. Die Größe des Codes ist deshalb von Bedeutung, da mit der Komplexität nicht nur die Fehleranfälligkeit steigt, sondern auch die Gefahr, die Vertrauenswürdigkeit schwerer nachweisbar zu machen.

Der Presenter umfasst ca. 2500 Zeilen Code. Um den vertrauenswürdigen Teil so gering wie möglich zu gestalten, können ca. 500 Zeilen ausgelagert werden, d.h. einzelne Module externalisiert werden.

5.2 Bewertung der Performance

Um die Performance beim Darstellen der Präsentation zu messen, wurde die Latenzzeit beim Umschalten zwischen zwei Folien ausgewertet, sowie das Laden einer Präsentation bis zur ersten Darstellung.

Die Latenzzeit wurde gemessen, um zu bewerten, ob es ohne spürbare Verzögerungen möglich ist, das Umschalten zwischen zwei Folien zu ermöglichen. Beim Schalten auf die nächste Folie müssen einige komplexe Aufgaben gelöst werden. Es wird die PNG-Datei mit Hilfe der libpng und einer Dekompression durch die zlib in ein Rohdatenformat umgewandelt, die Daten müssen skaliert werden und in den VScreen-Puffer von DOpE kopiert werden.

Der zweite Punkt ist deshalb von Interesse, um abzuschätzen, in welchem Bereich die Wartezeit beim Laden von einer hohen Anzahl an Folien liegt.

Latenzzeit zwischen dem Umschalten von zwei Folien Um die Wartezeit zwischen dem Umschalten zweier Folien, d.h. zweier PNG-Dateien, wurde eine Zeitmessung in der Controller-Schicht des Presenters durchgeführt. Es wurde die Zeit vom Drücken eines Buttons bis zur Anzeige des nächsten Bildes in Mikrosekunden gemessen.

Untersucht wurde das Verhältnis der Größe einer PNG-Datei in kbyte und der Verzögerungszeit beim Umschalten auf die nächste Folie. Die Abbildung 9 lässt sich dahingehend interpretieren, dass die Latenzzeit beim Umschalten lediglich linear wächst. Außerdem zeigt sich, dass auch bei größeren Bildern die Wartezeit unter 1 Sekunde liegt. Damit ist selbst bei meinem gewählten älteren Testrechner keine merkbare Verzögerungszeit beim Umschalten der Folien zu erkennen.

Latenzzeit beim Laden einer Präsentation Für die Messung der Ladezeit einer Präsentation wurde vom Aufruf des File-Provider über das Einlesen einer die PNG-Dateien enthaltenden Konfigurationsdatei bis hin zum Darstellen der ersten Folie die Zeit gemessen.

Es wurde das Verhältnis Latenzzeit zur Anzahl der zu ladenden Folien untersucht. Um die Gesamtgröße vergleichbar zu halten, besteht jede Präsentation aus der wachsenden Anzahl einer Testfolie. Sie hat die Größe von 138kbyte. Das Ergebnis der Untersuchung war, das eine Verdopplung der Folienanzahl eine Verdopplung der Ladezeit bedeutet, d.h. ebenfalls ein linearer Anstieg.

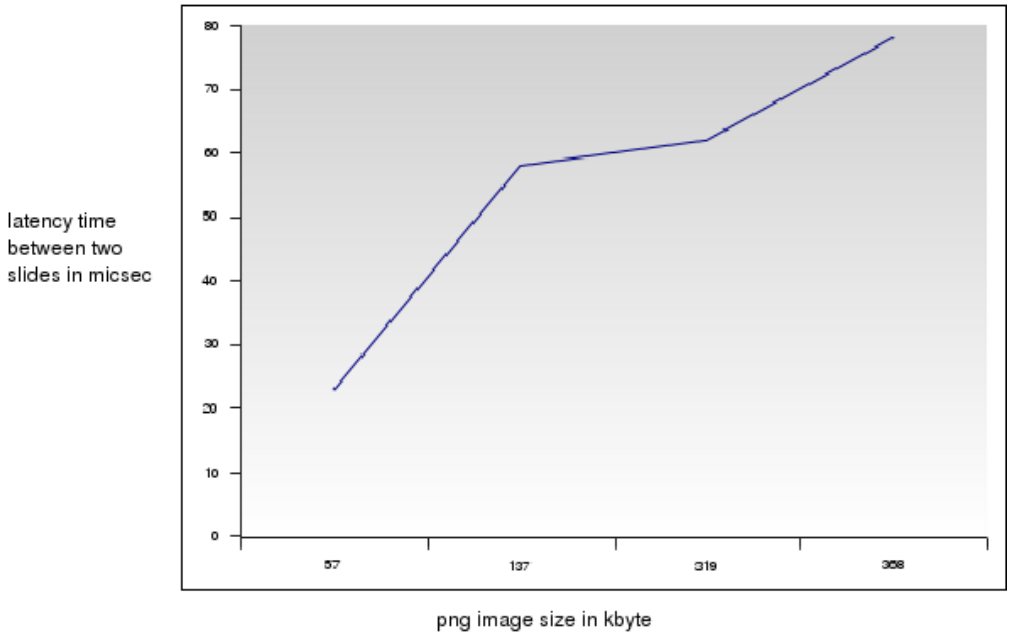

Abbildung 9: Latenzzeit zwischen dem Umschalten zweier Folien bzgl. der Größe des Bildes in kbyte und der Zeit in Mikrosekunden

5.3 Speicherverbrauch der Präsentation

Für den Benutzer des Presenter ist es wichtig zu wissen, wie viele Folien maximal eine Präsentation enthalten darf bei einer bestimmten Menge an Arbeitsspeicher, um die Software praxistauglich einsetzen zu können.

Für die Bewertung hat die Größe des Speichers, die der Dataspace-Manager verwaltet, eine besondere Bedeutung. Da der Presenter die Folien als Dataspaces verwaltet, hängt die maximale Anzahl der Folien von dieser Größe ab. Je mehr L4 Server mit dem Presenter gestartet werden, die sich Speicher vom Dataspace-Manager allokieren, umso kleiner wird die maximal mögliche Folienanzahl.

In der aktuellen Version des Presenters werden alle Folien am Anfang eingeladen, damit daraus die Vorschau-Bilder generiert werden können. Danach werden alle Dataspaces im Hauptspeicher gehalten. Obwohl der gewählte Ansatz den meisten Hauptspeicherbedarf gegenüber anderen Lösungen hat, bringt er eine sehr kurze Latenzzeit beim Umschalten der Folien, da die meiste Zeit, wie oben untersucht, für das initiale Laden der Folien verbraucht wird. Wenn aber der Speicher, der dem Dataspace-Manager zur Verwaltung gegeben wird, sehr klein ist, verringert sich dementsprechend die maximale Folienanzahl.

Eine Lösung für das Problem ist die Einführung eines Working Sets. Darauf möchte ich im Ausblick noch einmal eingehen.

6 Zusammenfassung und Ausblick

Im Rahmen dieses Beleges wurde eine Architektur entworfen, die es ermöglicht, bewährte Linux Softwarelösungen, die keiner Kontrolle auf Vertrauenswürdigkeit und Integrität unterlegen haben, in einer vertrauenswürdigen Mikrokern-Umgebung wiederzuverwenden. Verschiedene Szenarien, so u.a. auch DRM, wurden auf ihre Interaktion und Auswirkungen in Bezug auf Sicherheitsaspekte in Zusammenhang mit dem Konzept untersucht. Als Proof-Of-Concept wurde der Presenter entwickelt, ein L4 Server zur Darstellung von Präsentationen.

Um die Architektur real auf eine sichere Plattform zu stellen, wird ein TPM benötigt, um die bis jetzt nur abstrakt vorhandene Verankerung in der Hardware und die Integrität bzw. Vertrauenswürdigkeit der L4 Server zu garantieren.

Als ein Problem wurde die Integrität der entschlüsselten Daten in einem DRM-Szenario angesprochen. Für eine reale Implementierung inklusive dem notwendigen Hashcode vom Content Provider ist eine nähere Untersuchung notwendig, um weitere Angriffe auszuschließen.

Um nicht wie in der aktuellen Version alle Folien im Hauptspeicher zu halten, ist als eine Erweiterung für den Presenter die Implementierung eines Working Sets vorgesehen. Ein Working Set ist die Menge von Folien, die gerade aktuell im Arbeitsspeicher gehalten wird. Durch eine dynamische Ersetzung der vorhandenen ArrayList können sehr flexibel unterschiedliche Strategien zur Aktualisierung des Working Sets umgesetzt werden. Denkbar sind im nächsten Schritt zwei neue ArrayList-Implementierungen, eine für binäre, eine zweite für Konfigurationsdateien, die einen direkten Pfad auf die PNG-Dateien enthalten.

Der Presenter sollte dahingehend untersucht werden, ob weitere Funktionsauslagerungen zur Minimierung des vertraulichen Codes möglich sind. Es bietet sich z.B. an, die Skalierungsfunktion und die Konvertierung der PNG-Bilder in das Format von DOpE ebenfalls von convert durchführen zu lassen.

Außerdem lohnt die Diskussion einer Einbringung des File-Providers und der execv-Schnittstelle in die dietlibc, die kontinuierlich für L4 weiterentwickelt wird. Eine überlegenswerter Ansatz wäre die Implementierung eines Backend, der Dataspaces über das read/open/write-Interface verfügbar macht und auf den File-Provider zugreift.

7 Glossar

BMP Bitmap Picture, Bildformat, das ursprünglich für Windows entwickelt wurde

Dataspace Speicherobjekt, das u.a. Rechte anbietet

DOpE Desktop Operating Environment, Fenstersystem für das Echtzeitbetriebssystem DROPS

DRM Digital Rights Managment, System zum Schutz hochwertiger digitaler

Medien, der Verteilung und Nutzung

DROPS Dresden Realtime OPerating System

IDL Interface Definition Language

L4Env User Level Umgebung für L4 Systeme, die mehrere Server enthält, z.B. Loader und DM_Phys

L4Linux Linux als eigener Server auf L4, bietet eine Unix-ähnliche Schnittstelle für L4

PNG Portable Network Graphic, lizenzfreies Bildformat, ermöglicht verlustfreie Kompression und bietet einen Alphakanal

Literatur

[1] L4 Loader website. URL:
`http://os/~fm3/doc/loader/`.

[2] libpng website. URL:
`http://www.libpng.org/pub/png/libpng.html`.

[3] TCPA website. URL:
`http://www.trustedcomputing.org`.

[4] M. Aron, J. Liedtke, K. Elphinstone, Y. Park, T. Jaeger, and L. Deller. The SawMill framework for virtual memory diversity. In *Australasian Computer Systems Architecture Conference, Goldcoast, Queensland, Australia*, pages 3–11. IEEE Computer Society Press, 2000.

[5] Joan Feigenbaum, Michael J. Freedman, Tomas Sander, and Adam Shostack. Privacy engineering for digital rights management systems. In *Proceedings of the ACM Workshop in Security and Privacy in Digital Rights Management*, November 2001.

[6] Norman Feske and Hermann Härtig. DOpE - a window server for real-time and embedded systems. Technical Report TUD-FI03-10, TU Dresden, 2003.

[7] Hermann Härtig. Security architectures revisited. In *Proceedings of the 10th ACM SIGOPS European Workshop*, September 2002.

[8] Andreas Pfitzmann. Sicherheit in Rechnernetzen: mehrseitige Sicherheit in verteilten und durch verteilte Systeme, 2000.