Die Nizza-Systemarchitektur

Felix Neumann
Technische Universität Ilmenau
felix.neumann@tu-ilmenau.de

3. Februar 2009

Zusammenfassung

Die Zahl stetig neu entdeckter Sicherheitslücken in verbreiteten Betriebssystemen zeigt einen klaren Bedarf an sicheren Systemen auf. Die Nizza-Systemarchitektur, welche in dieser Arbeit vorgestellt werden soll, bietet einen Weg, die Vertrauenswürdigkeit vorhandener Software zu erhöhen.

Hierzu werden solche Softwarekomponenten, denen zur Herstellung bestimmter Sicherheitseigenschaften vertraut werden muss, streng isoliert und mit geringer Komplexität ausgestattet. Zugleich wird möglichst viel Software wiederverwendet, was hohe Funktionalität und geringen Portierungsaufwand gewährleisten soll. Techniken zur entfernten Authentifizierung des Systems zielen darauf ab, seine Vertrauenswürdigkeit zusätzlich zu steigern.

Diese Arbeit beschreibt den Aufbau der Architektur beginnt dabei auf den ihr zugrundeliegenden Prinzipien und Technologien. Im Anschluss präsentiert sie zwei auf der Basis Nizzas realisierte Anwendungen, um einen Eindruck von der Praxistauglichkeit der Architektur zu vermitteln.

1 Einführung

Auf Handys, mobilen Geräten und Desktopcomputern werden heutzutage eine vielzahl unterschiedlicher Anwendungen ausgeführt, welche verschiedenste Anforderungen an das sie beherrbergende Computersystem stellen. So wird die Nutzung klassischer Bürosoftware, wie z.B. Textverarbeitung und Tabellenkalkulation, nicht selten mit sicherheitskritischen Applikationen wie Homebanking oder verschlüsselter Kommunikation verknüpft.

Einige häufig wiederkehrende Anforderungen, die moderne Betriebssysteme erfüllen müssen, sind dabei [HHF+05, Kap. 1]:

- **Universelle Nutzbarkeit**; das heißt, Anwendungen müssen ein möglichst breites Spektrum an Funktionalität implementieren können. Voraussetzung hierfür ist auch ein reiches Angebot an Diensten, welche dann von mehreren Applikationen nutzbar sind – Beispiele sind eine grafische Nutzeroberfläche oder Netzwerkfunktionalität.

- **Sicherheit**. Nicht wenige Applikationen müssen diverse Sicherheitseigenschaften herstellen und fordern selbst derartige Eigenschaften vom sie beherrbergenden System. So soll häufig der vertrauliche und integere Umgang mit Informationen gewährleistet werden.

  Das Durchsetzen jeglicher Sicherheitseigenschaften bedarf einiger Voraussetzungen, die das Betriebssystem schaffen muss. Hier sei insbesondere die Isolation der einzelnen Prozesse untereinander genannt.

Gehört zu einer Applikation, wie beispielsweise das Homebanking, eine Sicherheitsanforderung, wie die vertrauliche und integere Kommunikation mit dem Bankserver, so bezeichnet nach [HHF+05, Kap. 1] die „*Trusted Computing Base*“ (TCB) diejenige Menge an Software, welche die Anwendung unbedingt benötigt – genauer: welche der Benutzer *vertrauen* muss –, um die jeweiligen Sicherheitseigenschaften sowie die volle Funktionalität zu gewährleisten.

Vertrauen setzt, ebenfalls nach [HHF+05, Kap. 2], voraus, dass die Software bestimmte funktionale, zeitliche und sicherheitsrelevante Anforderungen erfüllt.

Ein großes Problem bei heute üblichen Betriebssystemen ist dabei eine zu große TCB. Beispielsweise sorgen beim Linux-Betriebssystem mangelnde Isolation und weit verstreute Sicherheitsmechanismen dafür, dass die Menge an Software, der Vertrauen geschenkt werden muss, schon auf Seite des Kernels mehr als eine Million Lines of Code umfasst [SPHH06, Tab. 5].

Eine derartig große Quelltextbasis macht eine formale Verifikation der Software unmöglich und bringt eine hohe Fehlerwahrscheinlichkeit mit sich. Demonstriert wird dies durch die enorme Zahl an Sicherheitslücken, die beständig in alltäglich eingesetzter Software entdeckt wird [MoK06]. Solche Betriebssysteme bieten also nur ungenügende Sicherheit.

Hier setzt die Nizza-Systemarchitektur an: ihr Ziel ist es, Anwendungen mit hohen Sicherheitsanforderungen eine vertrauenswürdige Plattform zu bieten, ohne dabei die Unterstützung für vorhandene Software einzubüßen.

Auf diese Art und Weise soll die Funktionalität und *universelle Nutzbarkeit* vorhandener Betriebssysteme mit der Vertrauenswürdigkeit einer Systemarchitektur vereinigt werden, die darauf zugeschnitten ist, hinreichende *Sicherheit* zu bieten.

In dieser Arbeit soll die Nizza-Architektur vorgestellt werden. Ihr Aufbau wird in Kapitel 2 beschrieben, hier finden sich auch Erläuterungen über grundlegende Entwurfsprinzipien und eingesetzte Technologien.

In Kapitel 3 werden zwei Anwendungen eingeführt, die auf Grundlage von Nizza realisiert wurden. Dieses Kapitel bietet zugleich Raum für eine Bewertung der Architektur.

Schließlich enthält das 4. Kapitel eine Zusammenfassung der Ergebnisse der Arbeit.

2 Architektur

Bei Nizza handelt es sich um eine Architektur für sichere Systeme, deren Struktur und Komponenten dieses Kapitel nahebringen soll. Hierzu beschreibt der Abschnitt 2.1 einleitend grundlegende Prinzipien und Strategien, anhand derer die Architektur später begründet und nachvollzogen werden kann. Abschnitt 2.2 führt Technologien ein, welche in Nizza angewandt werden.

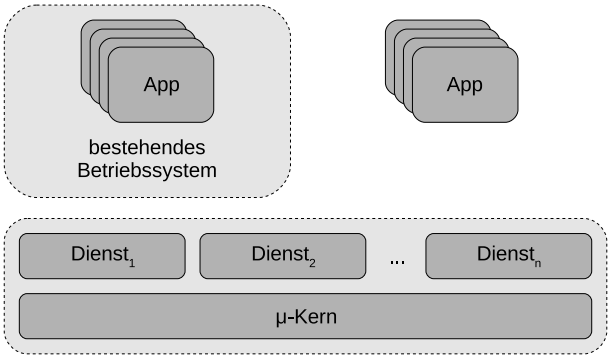

Abbildung 1: Die Komponenten der Nizza-Systemarchitektur.

Nizzas Architektur besteht aus drei Schichten, wie in Abbildung 1 dargestellt. In der untersten Schicht liegt der Betriebssystemkern; Erläuterungen finden sich in Abschnitt 2.3. Die folgende Ebene enthält eine Reihe vertrauenswürdiger Dienste, welche für Anwendungen bereitstehen. Diese sog. „Systemplattform“ wird in Abschnitt 2.4 erklärt. Abschnitt 2.5 enthält Informa-

tionen zu Aufbau und Besonderheiten der obersten Schicht; in ihr laufen die Anwendungen ab. Einige von ihnen werden in einem virtualisierten Betriebssystem ausgeführt – hierzu berichtet der selbe Abschnitt.

2.1 Zugrundeliegende Prinzipien

Ausgangspunkt zur Darlegung der Prinzipien sind die bereits in der Einleitung aufgeführten Anforderungen an Nizza, nämlich *Vertrauenswürdigkeit* als Plattform für sicherheitskritische Anwendungen, sowie *Funktionalität* beziehungsweise universelle Nutzbarkeit.

Aus diesen Anforderungen heraus lassen sich die folgenden Prinzipien ableiten, welche ihrerseits Grundlage für Betrachtungen über die Nizza-Architektur selbst sind.

Eine kleine TCB mit isolierten Komponenten. [HHF+05, Kap. 2] stellt fest, dass das Herstellen von Sicherheitsanforderungen oftmals nur einen kleinen Teil der Gesamtfunktionalität einer Anwendung ausmacht. Ein Großteil der Software kümmert sich demnach um Aufgaben wie Ein-/Ausgabe, Berechnungen, Kommunikation, Benutzerführung und ähnliches, welche – für sich genommen – keinen Einfluss auf die Sicherheitseigenschaften haben.

Isoliert man also jenen Teil der Anwendung, dem vertraut werden muss, und sorgt für kontrollierte Kommunikation zwischen beiden Parts, fällt das Herstellen von vertrauensrelevanten Eigenschaften wie auch der Nachweis hierüber wesentlich leichter. Dies ist nachvollziehbar: kleine Software mit kontrollierten Wechselwirkungen nach außen hin ist leichter zu handhaben und leichter formal zu verifizieren als sehr komplexe Software mit nicht klar definierten Wechselwirkungen und Seiteneffekten.

Auf diese Weise versucht Nizza, alle Komponenten einer Anwendungs-TCB klein und isoliert zu halten. Dies ist äquivalent zur in [Här02, Kap. 3.1] erwähnten „*Technik der kleinen Schnittstellen*“.

Doch nicht nur die Bestandteile der TCB sollen möglichst klein gehalten werden. Auch die TCB selbst soll aus so wenigen Komponenten wie möglich bestehen und wird daher für jede konkrete Anwendung individuell zusammenzustellt. Durch dieses Prinzip wird Vertrauenswürdigkeit zu einer kontrollier- und nachweisbaren Eigenschaft.

Wiederverwendung unvertrauter Komponenten. Wie in [SPHH06, Kap. 2.1] dargelegt ist es Ziel der Entwickler von Nizza, die volle Funktionalität eines gewöhnlichen Betriebssystems zu gewährleisten. Das heißt für Nizza: Benutzer sollen die volle, für sie gewohnte Funktionalität wiederfinden

können. Zum anderen sollen auch aus Anwendungssicht all jene Softwarekomponenten verfügbar sein, die sich auch in üblichen Betriebssystemen finden lassen. Hierzu gehört beispielsweise der Netzwerkstack oder ein Dateisystem.

Das Prinzip der Wiederverwendung wird konsequent in fast allen Punkten der Nizza-Architektur eingesetzt; Beispiele sind das vertrauenswürdige Dateisystem (Abschnitt 2.4) oder die Virtualisierung eines ganzen Betriebssystems (Abschnitt 2.5).

2.2 Eingesetzte Technologien

Nizzas Entwickler setzen in ihrer Architektur einige Techniken ein, welche dieses Kapitel erläutern soll. Es handelt sich insbesondere um *Trusted Wrappers* sowie das *Authenticated Booting*.

2.2.1 Trusted Wrappers

Um das Prinzip der Wiederverwendung bestehender Software durchsetzen zu können, ohne dabei das Erreichen der Sicherheitseigenschaften zu gefährden, verwenden die Entwickler der Nizza-Architektur die sogenannten Trusted Wrappers [HHF+05, Kap. 2]. Dabei wahren vertraute Systemkomponenten in höheren Schichten Vertraulichkeit und Integrität, sodass in tieferen Architekturebenen problemlos unvertraute Software eingesetzt werden kann.

Typische Szenarien zur Verwendung der Trusted Wrappers ist der Einsatz eines vorhandenen Netzwerkstacks oder Dateisystems, an welche Informationen nur verschlüsselt und signiert weitergegeben werden. Auch die in Kapitel 3 vorgestellten Anwendungen setzen Trusted Wrappers intensiv ein.

2.2.2 Authenticated Booting

Die Konstruktion einer Vertrauen schaffenden Architektur nützt nichts, wenn Angreifer Komponenten der Architektur modifizieren können. Um dies zu vermeiden muss eine Möglichkeit gefunden werden, die Echtheit des Betriebssystems zu jedem Zeitpunkt überprüfen zu können.

Hierzu verwendet Nizza das *Trusted Platform Module* (TPM) der Trusted Computing Group [TPM09]. Es handelt sich, wie in [HHF+05, Kap. 6] beschrieben, um einen Hardwarebaustein, der erstens über einen geschützten Speicher verfügt und es zweitens gestattet, die eingesetzte Software von einem entfernten Computersystem aus zu überprüfen.

Beide Funktionalitäten werden im folgenden vorgestellt. Vertrauensgrundlage ist in jedem Fall das korrekte Funktionieren des Trusted Platform Modules sowie die Sicherheit des eingebauten Speichers.

Mit TPM lässt sich zur Laufzeit von einem entfernten Computersystem aus feststellen, ob auf dem lokalen System eine bestimmte Anwendung auf einem bestimmten Betriebssystem abläuft (*Remote Attestation*). Hierzu bildet das Trusted Platform Module beim Systemstart eine digital signierte Prüfsumme über das geladene Betriebssystem, die über ein Challenge-Response-Protokoll entfernt abgerufen werden kann.

Das Betriebssystem kann selbst digitale Prüfsummen hinzufügen, um so auch Anwendungen testieren zu können.

Außerdem ist in TPM ein *Sealed Memory* genannter, geschützter Speicher enthalten, der es einem Betriebssystem erlaubt, eigene Schlüssel darin abzulegen. Dies gestattet beispielsweise Verschlüsselung und Signierung von Informationen auf nicht vertrauenswürdigen Medien wie Dateisystemen (siehe Abschnitt 2.4). Dabei ist die Zugriffsbefugnis von der Systemkonfiguration abhängig: Schlüssel, die ein Betriebssystem im geschützten Speicher abgelegt hat, kann nur das selbe System wieder abrufen.

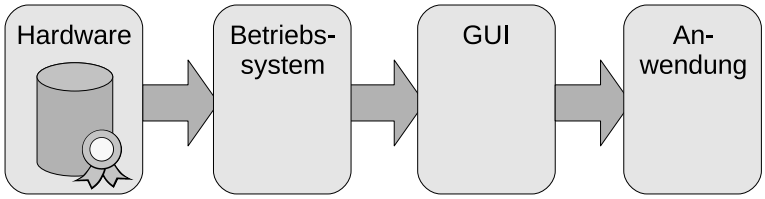

Abbildung 2: Die Vertrauenskette. Die Hardware, welche einen sicheren Speicher enthält, führt einen Nachweis über die Identität des Betriebssystems, dieses weist die Identität der grafischen Oberfläche nach; die grafische Oberfläche bietet Informationen zur Identität der angezeigten Anwendung.

Mit diesen Mitteln ist es nach [Här02, Kap. 3.3] möglich, eine Vertrauenskette zum Anwender des Systems aufzubauen, wie in Abbildung 2 dargestellt. Dabei kann sich der Benutzer versichern, dass die gewünschte Anwendung unverändert auf dem gewünschten System abläuft. Abschnitt 2.4 enthält nähere Informationen zur Umsetzung dieser Vertrauenskette in Nizza.

2.3 Der Mikrokern

Auf der untersten Schicht der Nizza-Systemarchitektur steht der Betriebssystemkern. Er verwaltet die wichtigsten Betriebsmittel des Computers und stellt dabei Eigenschaften her, die wesentlich für fast alle nichtfunktionalen

Eigenschaften des Systems sind. Der Kern ist also unumgänglich Bestandteil der TCB, und damit – gemäß den in Abschnitt 2.1 dargelegten Prinzipien – so klein wie möglich zu halten: ein Mikrokern.

Eine in besonderem Maße sicherheitsrelevante Eigenschaft ist das Herstellen von Isolation, worauf fast alle weiteren Sicherheitseigenschaften beruhen. [HHF+05, Kap. 4.1] unterteilt Softwarekomponenten dabei in Schutzdomänen, die der Systemkern voneinander isolieren muss. Zusätzlich bedarf es einer kontrollierten Möglichkeit der schnellen Kommunikation zwischen diesen Domänen.

Nizza verwendet den *Fiasco*-Mikrokern ([HH01]), welcher an der TU Dresden im Umfeld der Echtzeitbetriebssysteme entstand und alle oben genannten Anforderungen erfüllt. Fiasco ist vollkommen kompatibel zum *L4 microkernel interface* [Lie95], welches mit einer sehr geringe Zahl von Systemaufrufen auskommt und möglichst viel Funktionalität in den Userspace verlagert.

Gründe für den Einsatz der L4-Kernelschnittstelle sind:

- **Geringer Umfang.** [HHF+05, Kap. 4.2] führt auf, dass L4-Kerne lediglich drei Abstraktionen an ihrer Schnittstelle anbieten: Adressräume, Threads sowie Interprozesskommunikation (IPC) (siehe auch [HHL+97, Kap. 3]). Weitere Funktionen, die das Betriebssystem zur Verfügung stellen soll, werden im Userspace implementiert; hierzu zählen Speicherverwaltung (einschließlich Paging), Gerätemanagement, Interruptbehandlung und vieles mehr. (In der Nizza-Architektur sind solche Dienste in der Systemplattform, siehe Abschnitt 2.4, zusammengefasst.)

  Das hier erkennbare Architekturmuster gestattet eine stark ausgeprägte Trennung der Betriebssystemfunktionalität in kleine, isolierte Komponenten, was den in 2.1 genannten Prinzipien entgegenkommt. Darüber hinaus gestattet es viel Flexibilität bei der Wahl der Implementierung vieler Betriebssystemfunktionen.

- **Austauschbare Implementierung, Portierbarkeit.** Es existieren mehrere Implementierungen der L4-Schnittstelle für verschiedene Plattformen und mit unterschiedlich ausgeprägten nichtfunktionalen Anforderungen.

  So gibt es Versionen für x86, ARM, Alpha und MIPS ([HHF+05, Kap. 4.2], [HHL+97, Kap. 3]). Die urpsrüngliche x86-Implementierung, detailiert beschrieben in [Lie95], verwendete einige Tricks um architekturspezifische Merkmale auszureizen; Fiasco wiederrum legt Wert auf lock-freie synchronisierung zentraler Datenstrukturen [HH01] und Echtzeiteigenschaften [Här07].

Die Austauschbarkeit der Implementierung bietet viel Flexibilität in Bezug auf zukünftige Entwicklungen.

Der Einsatz des Fiasco-Kern in Nizza wird durch folgende Eigenschaften begünstigt:

- **Geringer Umfang.** Fiasco wurde in weniger als 15.000 Lines of Code implementiert. Dies entspricht der Vorgabe, nur kleine und damit überschaubare und weniger fehlerträchtige Komponenten in die TCB zu integrieren.
- **Kontrolle.** Der Fiasco-Mikrokern wurde an der TU Dresden entwickelt und ist somit besser durch die Nizza-Entwickler kontrollierbar. So nennt [HHF+05] geplante Erweiterungen der Mikrokern-Schnittstelle hinsichtlich seiner Sicherheitseigenschaften, nämlich Ressourcen- und IPC-Kontrolle.
- **Verifizierbarkeit.** Ein Projekt an der TU Dresden beschäftigt sich mit der Verifizierung des Fiasco-Quelltextes. Dies unterstützt das Sicherheitsziel der Nizza-Architektur.

2.4 Die Systemplattform

Dienste und Funktionalitäten, die den Anwendungen des Systems zur Verfügung stehen, jedoch nicht im Kern implementiert werden sollen, laufen in Form gewöhnlicher Anwendungsprozesse ab. Nizzas Entwickler fassen sie in [HHF+05, Kap. 4.1] zur „Systemplattform“ zusammen, [SPHH06, Kap. 2.2.2] gruppiert sie zur Ausführungsumgebung der Anwendungen und prägt den Namen *L4Env*. Allen Systemplattformdiensten gemein ist ihre Vertrauenswürdigkeit, aufgrund derer sie Bestandteil der TCB werden können.

Im Folgenden sollen einige der Systemplattformdienste vorgestellt werden. Darüber hinaus befinden sich auf dieser Schicht der Architektur Gerätetreiber, Server zur Ressourcenverwaltung, Objektmanager und vieles mehr.

Der *Loader*. Beim Loader handelt es sich nach [HHF+05, Kap. 4.1] um einen zentralen Dienst, welcher vertraute Komponenten lädt und im System einrichtet. Dies umfasst neben dem Durchsetzen von Zugriffskontrolle auch die Unterstützung von *Authenticated Booting*, wobei eine Vertrauenskette von der Hardware bis hin zur Anwendung aufgebaut wird. Die Technik des Authenticated Booting wird in Abschnitt 2.2.2 näher beschrieben.

Durch seine vertrauensbildende Rolle trägt der Loader direkt zum Sicherheitsziel der Nizza-Architektur bei.

Die *trusted GUI*. Nizza bietet eine *Nitpicker* genannte, vertrauensbildende grafische Oberfläche an [HHF+05, Kap. 4.4], welche die Wahrung der in 2.1 erläuterten Prinzipien unterstützt.

Nitpicker sichert die Isolation der Anwendungen untereinander zu, indem es ihnen getrennte, virtuelle Framebuffers zur Bildschirmdarstellung zur Verfügung stellt. Auf diese Art und Weise wird verhindert, dass eine Anwendung die Bildschirmausgabe einer anderen beeinflussen oder auslesen kann. Die einzelnen virtuellen Ausgabebereiche werden durch Nitpicker zum physischen Framebuffer zusammengesetzt.

Gleichermaßen leitet Nitpicker Benutzereingaben, welche beispielsweise über die Tastatur erfolgten, nur zu genau einer Anwendung weiter, sodass keine zweite Applikation darauf Zugriff hat.

Auch aus Benutzersicht stellt Nitpicker eine klare Trennung der Anwendungen sicher: nach [SPHH06, Kap. 2.2.2] dunkelt die GUI solche Anwendungen, die nicht fokussiert sind, ab. [HHF+05, Kap. 4.4] erwähnt, dass jede Anwendung mit ihrem Namen ausgewiesen wird. Der Name selbst stammt dabei nicht von der Anwendung, sondern wird vom Loader ermittelt, sodass der Benutzer einen unverfälschbaren Beweis über die Identität der Applikation, welche er bedient, erhält.

Zum zweiten Prinzip, der Wiederverwendung vorhandener Software, stellt [HHF+05, Kap. 4.4] fest, dass sich durch Nitpicker kaum Einschränkungen ergeben. Nicht für Nizza ausgelegte grafische Nutzeroberflächen, selbst ganze Desktopumgebungen, können so problemlos wiederverwendet werden.

Nitpicker bietet derzeit keine Unterstützung für Hardwarebeschleunigung, deren Absicherung, so das genannte Paper, noch Gegenstand der Forschung ist. Probleme ergeben sich hier durch mangelnde Möglichkeiten zur Isolation der Anwendungen. Die Konstruktion eines einheitlichen Wrappers wird erschwert, da viele Grafikkarten unterschiedliche Schnittstellen ausweisen.

Vertrautes Dateisystem. Wie eingangs erwähnt, finden sich die Vertraulichkeit und die Integrität der im System befindlichen Daten unter den Zielen der Nizza-Architektur. Möchte man Informationen nun permanent abspeichern, muss garantiert sein, dass die Sicherheitseigenschaften auch bei physischem Zugriff des Angreifers auf das Speichermedium aufrecht erhalten werden.

Auch hier lässt sich das Prinzip der Wiederverwendung anwenden. Mithilfe eines Trusted Wrappers (siehe 2.2.1) können Informationen vertraulich und integer auf dem Dateisystem eines unvertrauten Betriebssystems abgelegt werden. Einen derartigen Ansatz erläutert [WH08] näher, die Implementierung gelang in 5.000 Lines of Code.

Voraussetzung zur Implementierung eines solchen sicheren Dateisystems ist ein kleiner, sicherer Speicher, der nur von authentifizierter Software abgefragt werden kann. Solch ein Speicher steht beispielsweise in einem *Trusted Platform Module* [TPM09] zur Verfügung – näheres erläuterte bereits Abschnitt 2.2.2.

Notwendig zur erfolgreichen Wahrung von Vertraulichkeit und Integrität ist nunmehr die Korrektheit und Unangreifbarkeit des sicheren Speichers. Möchte man weitere Eigenschaften garantieren, wie Wiederherstellbar- oder gar Verfügbarkeit der Informationen, bedarf es einer wesentlich größeren TCB, wie [HHF+05, Kap. 4.5] ausführt.

2.5 Die Anwendungsschicht

Oberhalb der Systemplattform liegt in der Nizza-Architektur die Schicht der Anwendungsprozesse. Mit dem Prinzip im Hinterkopf, vorhandene Software wiederverwenden zu können, statteten Nizzas Entwickler die Architektur mit der Fähigkeit aus, Anwendungen eines bestehenden Betriebssystems unverändert auf dem System auszuführen. Diese unvertrauten Anwendungen dürfen dabei niemals den vertrauenswürdigen Applikationen schaden [Här02, Kap. 4.1].

Um solch ein Konzept umzusetzen, muss das gesamte Betriebssystem virtualisiert werden [HHF+05, Kap. 4.3]. Nizzas Entwickler entschieden sich hier für die Paravirtualisierung eines Linux-Kerns. Dabei wird der Betriebssystemkern so verändert, dass er nicht mehr direkt auf die Hardware zugreift, sondern die Schnittstelle des Mikrokerns verwendet. Die Wahl des Linux-Betriebssystems begründet sich zum einen in seiner Beliebtheit und Funktionalität. Zum anderen ist Linux quelloffen, was zur Paravirtualisierung zwingend erforderlich ist.

[HHL+97] geht näher auf den *L⁴Linux* genannten, abgeänderten Linux-Kern ein. Die dort aufgeführten Benchmarks zeigen, dass Anwendungen auf dem paravirtualisierten System etwa 5 bis 10 Prozent an Geschwindigkeit einbüßen.

Soll eine sicherheitssensible Linux-Applikation mit einer möglichst kleinen TCB auskommen, muss diese modifziert werden, sodass ihre sicherheitskritischen Komponenten unabhängig vom unvertrauten L⁴Linux ablaufen können. Dieser *AppCore* genannte Teil der Anwendung vertraut dann nur noch ausgewählten Teilen der Systemplattform, dem Mikrokern sowie der unterliegenden Hardware.

Das Extrahieren dieses AppCores aus dem Quelltext – [HHL+97, Kap. 3.1] beschreibt den Vorgang genau – ist dabei je nach Anwendung unter-

schiedlich aufwändig. Zuerst sind die sicherheitskritischen Teile zu identifizieren, anschließend müssen sie extrahiert und in einem AppCore vereinigt werden. In einem letzten Schritt stellt der Entwickler die Kommunikation zwischen AppCore und der urpsrünglichen Anwendung her, hierzu lassen sich beispielsweise vorhandene Schnittstellen wiederverwenden und Lücken mit Stubs auffüllen.

Die Architekten Nizzas weisen dabei auf zwei Probleme hin. Erstens hängt der Portierungsaufwand stark von der Dokumentierheit und Wartbarkeit der Anwendung ab. Zweitens muss der Quelltext des AppCores möglicherweise modifiziert werden, um das Prinzip der geringen Softwarekomplexität zu wahren. Dies birgt jedoch das Problem der geringeren Funktionalität und der erhöhten Kosten zur Verbindung beider Anwendungsteile mit sich. Hier muss der Entwickler von Fall zu Fall abwägen.

Gerade der hohe Änderungsaufwand sowie die verminderte Funktionalität sind Kritikpunkte an der Nizza-Architektur [TMLL06].

3 Anwendungen

Um die Wirkung sowie Vor- und Nachteile der Nizza-Architektur zu veranschaulichen, präsentiert dieses Kapitel zwei Anwendungen, welche Wissenschaftler der TU Dresden auf Nizza implementiert haben. Die entsprechende Beschreibung findet sich bei [HHF+05, Kap. 5], [SPHH06, Kap. 3 und 4] sowie [HWF05].

Bei den beiden Anwendungen handelt es sich zum einen um das digitale Signieren von E-Mails, beschrieben in Abschnitt 3.1. Zum anderen betrachtet Abschnitt 3.2 ein VPN-Gateway, dessen Nizza-Implementierung *Mikro-SINA* genannt wird.

Beide Abschnitte stellen zunächst das Szenario vor, analysieren im Anschluss, welche Maßnahmen zum Portieren der Anwendung zu treffen sind und stellen in einem dritten Schritt verschiedene Metriken vor, anhand derer der Erfolg der Portierung gemessen werden soll.

In den Quellen wird dabei neben diversen Performanzmaßen und den *Source Lines of Code* (LOC) auch *McCabe's complexity metric* (MCC) angegeben. Letzeres ist ein Komplexitätsmaß, bei dem zu jeder Funktion im Quelltext die Zahl der unterschiedlichen Ausführungspfade gemessen wird.

3.1 Signieren von E-Mails

Szenario. Möchte ein Benutzer eine E-Mail versenden und dabei die Integrität ihres Inhaltes sicherstellen, kann er sie *digital signieren*. Hierzu benötigt

er neben der entsprechenden Software einen (mitunter passwortgeschützten) privaten Schlüssel.

Mozilla Thunderbird, ein quelloffener E-Mail-Klient, lässt sich um die entsprechende Funktionalität erweitern, hierzu kann das *Enigmail*-Plugin installiert werden. Dieses wiederrum verwendet die *GnuPG*-Bibliothek, welche unter anderem das digitale Signieren von Daten gestattet.

Der Vorgang läuft wie folgt ab. Vor dem Absenden einer E-Mail wählt der Benutzer die entsprechende Funktion in Thunderbird an und aktiviert damit das Enigmail-Plugin. Dieses fragt das Passwort zum privaten Schlüssel ab und gibt es, zusammen mit dem Inhalt der E-Mail, an GnuPG weiter. Die Bibliothek lädt dann den Schlüssel, unterzeichnet den E-Mail-Inhalt und sendet die Ergebnisse zurück an Enigmail.

Analyse. Um die sicherheitskritischen Teile der Applikation in einen AppCore zu extrahieren, müssen sie gemäß Abschnitt 2.5 zuerst identifiziert werden. Vertrauliche Daten sind der Schlüssel, das dazugehörige Passwort sowie der Inhalt der E-Mail.

Der Inhalt muss integer bleiben, daher sollte ihn der Benutzer vor dem Unterzeichnen über vertrauenswürdige Sotware einsehen können. Das Passwort sowie der (geöffnete) private Schlüssel sollen vertraulich gehalten werden und dürfen daher zu keinem Zeitpunkt nicht vertrauenswürdigen Systemkomponenten zugänglich sein.

Der AppCore realisiert also neben den entsprechenden Funktionen aus der GnuPG-Bibliothek auch ein kleines Benutzerinterface, welches den E-Mail-Inhalt im Plaintext anzeigt, den Benutzer zur Bestätigung auffordert und das Passwort entgegen nimmt.

Einen Überblick über das Systemsetup verschafft Abbildung 3.

Metriken. Der Umfang von Mozilla Thunderbird, Enigmail und GnuPG zusammen wird auf über 250.000 LOC beziffert. Zusätzlich beläuft sich ein Linux-Kernel mit XServer auf weitere 1.485.000 LOC. Die TCB des portierten AppCore hingegen umfasst, so die Entwickler, insgesamt 145.000 LOC, inklusive Fiasco-Mikrokern und Systemumgebung. (Der AppCore selbst enthält 54.000 LOC.)

Messungen gemäß dem MCC-Maß ergeben einen Wert von 45.000 für die Thunderbird-Applikation mit GnuPG; das Komplexitätsmaß für den AppCore beträgt 11.000.

Damit hat sich die Gesamtgröße der TCB um eine Größenordnung verkleinert, die Komplexität der Anwendung um den Faktor 4.

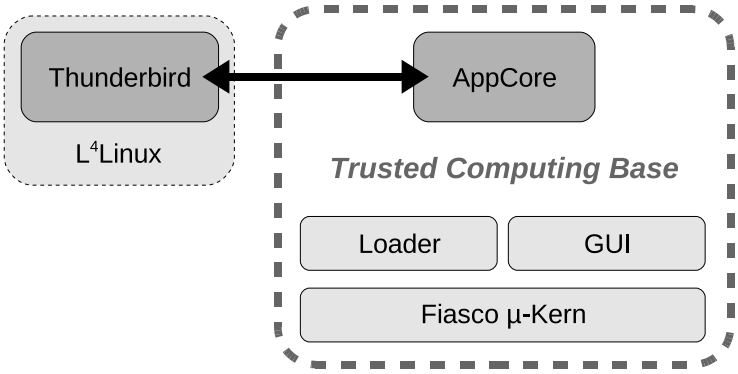

Abbildung 3: Die TCB der E-Mail-Applikation. Der AppCore enthält neben einer kleinen grafischen Oberfläche eine Portierung der GnuPG-Bibliothek. Die E-Mail-Anwendung selbst ist nicht Teil der hier grau umrandet dargestellten TCB und wird auf L^{4}Linux ausgeführt.

Bewertung. Die Größe der Trusted Computing Base und damit die Wahrscheinlichkeit für darin enthaltene Fehler ist durch das Portieren auf die Nizza-Architektur drastisch gesunken. Zum Portierungsaufwand finden sich bei den Entwicklern Nizzas nur sehr ungenaue Angaben.

Bekannt ist, dass weder der Quelltext von Thunderbird noch jener von Enigmail geändert werden mussten; es reicht, Enigmail zur Benutzung des AppCores zu konfigurieren. Über den Aufwand zur Portierung von GnuPG treffen die Entwickler in den genannten Quellen keine Angaben.

Die Funktionalität der Applikation blieb augenscheinlich erhalten, doch handelt es sich hier auch um einen eher einfachen Anwendungsfall (siehe Abschnitt 3.2). Messungen zur Performanz sind bei dem vorliegenden Szenario wenig sinnvoll.

3.2 VPN-Gateway

Szenario. Ein *virtuelles privates Netzwerk* (VPN) ist ein privates Netz, das über öffentliche Netze aufgebaut wird. Die Informationen werden unter Wahrung von Vertraulichkeit, Authentizität und Integrität durch ein öffentliches Netzwerk getunnelt. Die Grenze zwischen VPN-Teilnehmern und Internet bilden dabei *VPN-Gateways*; sie repräsentieren den Übergang zwischen vertrauenswürdigem und unvertrautem Netz und müssen für die Wahrung der genannten Sicherheitseigenschaften sorgen.

Die Entwickler Nizzas ziehen als Beispielimplementierung ein Gateway basierend auf einem Linux-Kern heran. Im Einsatz ist *Snapgear's Embed-*

ded Linux [Sna08], eine schmale Linux-Distribution für eingebettete Systeme; hinzu kommt die *FreeS/WAN*-Bibliothek [Fre04], eine quelloffene VPN-Implementierung, welche das IPsec-Protokoll einsetzt .

Im Falle von IPsec werden alle Botschaften, die ein Gateway aus dem lokalen Netz empfängt, nur verschlüsselt ins öffentliche Netzwerk weiterversendet; umgekehrt werden alle Informationen aus dem öffentlichen Netz entschlüsselt, bevor sie an den Empfänger im lokalen Netz weitergeleitet werden.

Analyse. Bei Gateways, die auf IPsec basieren und über einen monolithischen Linux-Kern verfügen, ist die IPsec-Komponente im Kernel implementiert und dort in den Netzwerkstack integriert. Die genannten Quellen stellen jedoch fest, dass die eigentlich sicherheitsrelevanten Funktionen nur circa 5 Prozent des Kerns ausmachen.

Die Nizza-basierende Implementierung eines VPN-Gateways verwendet einen AppCore, der lediglich jene Funktionen implementiert, die zur Herstellung der genannten Sicherheitseigenschaften relevant sind. Der AppCore baut dabei auf FreeS/WAN auf.

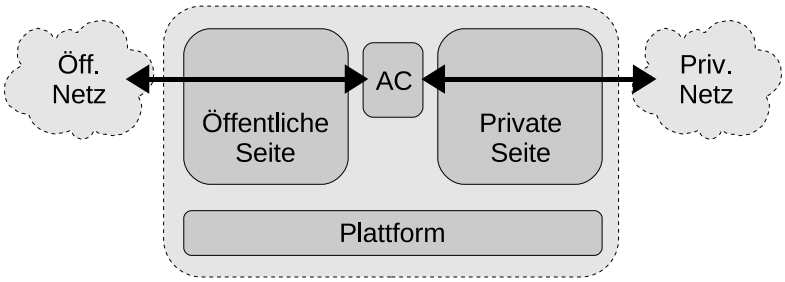

Abbildung 4: Die Architektur des VPN-Gateways. Die Trennung zwischen öffentlichem und privatem Netz spiegelt sich im Systemaufbau wieder. Öffentliche und private Seite, die beispielsweise über je ein L^{4}Linux implementiert werden können, implementieren Teile des Netzwerkstacks. Der AppCore trägt die sicherheitsrelevanten Verschlüsselungs- und Politikfunktionen.

Die Isolierung des privaten vom öffentlichen Netz spiegelt sich auch in der Gateway-Architektur wieder, welche in Abbildung 4 dargestellt ist. Beide physischen Netzwerkanschlüsse werden von je einer Systemkomponente bedient. *Private* wie auch *öffentliche Komponente* implementieren Teile des Netzwerkstacks und sind, abgesehen vom „Tor“ durch den AppCore hindurch, vollkommen voneinander isoliert. Auf diese Weise wird sichergestellt, dass keine Informationen aus dem privaten Netz ungeschützt in das öffentliche Netz gelangen.

Da die private Komponente unverschlüsselte Informationen sieht, ist sie Teil der TCB, ebenso wie der AppCore und die Systemplattform. Eine ursprüngliche Version der Mikro-SINA-Implementierung sah vor, auf privater wie auch öffentlicher Seite ein eigenständiges L^4Linux einzusetzen. Um die Möglichkeit von solchen Angriffen einzudämmen, welche auf die Verfügbarkeit des Systems abzielen, setzte man später auf beiden Seiten Teilimplementierungen des Netzwerkstacks ein. Die öffentliche Komponente benötigt für das IPsec-Protokoll einen kompletten TCP/IP-Stack, welcher als (unvertrauter) Prozess abläuft. Die private Komponente besteht aus einem Treiber für das Netzwerkinterface sowie einer Implementierung des Address Resolution Protocol (ARP), welches IP-Adressen in MAC-Adressen wandelt.

Metriken. Die beschriebene, auf Snapgear Linux aufbauende FreeS/WAN-Implementierung des Gateways beläuft sich auf etwa 155.000 LOC. Dem gegenüber stehen 74.000 LOC, welche das Mikro-SINA-Gateway umfasst. Sehr gering fallen hierbei die Anteile des AppCores (10.400 LOC) bzw. der FreeS/WAN-Implementierung (34.100 LOC) aus.

Die MCC-Komplexitätsmaße zeigen eine vergleichbare Minderung der Softwarekomplexität: 25.000 für das FreeS/WAN-Gateway, 10.000 für Mikro-SINA.

Zusammenfassend halbiert sich der Umfang des Gateways, während die Komplexität der IPsec-Komponenten sogar um ein drittel reduziert werden konnte.

Weniger positiv präsentieren sich Messungen zum Datendurchsatz, wie sie in [SPHH06, Tab. 9] angegeben wurde. Hier erreichte FreeS/WAN auf der Referenzmaschine einen Durchsatz von 63.8 Mbit/s; bei Mikro-SINA wurden lediglich 32.2 Mbit/s gemessen. Die Verfasser begründen diesen Verlust mit dem erhöhten Kommunikationsaufwand durch die Isolierung beider Seiten.

Bewertung. Zwar erreicht Mikro-SINA eine wesentlich bessere Isolierung des privaten und des öffentlichen Netzes und halbiert zusätzlich die Komplexität der TCB, jedoch geschieht dies stark auf Kosten der Performanz. Im konkreten Fall muss also zwischen Sicherheit und Geschwindigkeit abgewogen werden.

4 Schluss

Diese Arbeit stellte die Nizza-Systemarchitektur vor, deren Ziel es ist, die Trusted Computing Base sicherheitssensibler Anwendungen zu minimieren.

Um dies zu erreichen verwendet sie kleine, isolierte Komponenten und bestehende, unvertraute Softwarekomponenten. Nizza lässt das Betriebssystem, auf dem die jeweilige Anwendung basiert, paravirtualisiert auf einem Mikrokern ablaufen und isoliert schließlich sicherheitskritische Komponenten.

Auf diese Art und Weise konnte die TCB verschiedener Anwendungen um Faktor 2 bis 10 verkleinert werden, was die Software handhab- und zum Teil auch verifizierbar macht. Derartige Vorteile werden mit teilweise um die Hälfte verminderter Performanz sowie reduzierter Funktionalität bezahlt.

Literatur

[Fre04] *FreeS/WAN Project*. `http://www.freeswan.org`, 2004

[HH01] Hohmuth, Michael ; Härtig, Hermann: Pragmatic nonblocking synchronization for real-time systems. In: *Proceedings of the 2001 USENIX Annual Technical Conference (USENIX '01)*. Boston, MA, USA, Juni 2001

[HHF+05] Härtig, Hermann ; Hohmuth, Michael ; Feske, Norman ; Helmuth, Christian ; Lackorzynski, Adam ; Mehnert, Frank ; Peter, Michael: The Nizza Secure-System Architecture. In: *Proceedings of CollaborateCom 2005*. San Jose, CA, USA, 2005

[HHL+97] Härtig, Hermann ; Hohmuth, Michael ; Liedtke, Jochen ; Schönberg, Sebastian ; Wolter, Jean: The Performance of Microkernel based Systems. In: *16th ACM Symposium on Operating Systems Principles (SOSP'97)*. Saint-Malo, Frankreich, Oktober 1997

[HWF05] Helmuth, Christian ; Warg, Alexander ; Feske, Norman: Mikro-SINA – Hands-on Experiences with the Nizza Security Architecture. In: *Proceedings of the D.A.CH Security 2005*. Darmstadt, Deutschland, März 2005

[Här02] Härtig, Hermann: Security Architectures Revisited. In: *Proceedings of the 10th ACM SIGOPS European Workshop (EW 2002)*. Saint-Emilion, Frankreich, September 2002

[Här07] Härtig, Hermann: *Fiasco*. `http://www.inf.tu-dresden.de/index.php?node_id=1435`, 2007

[Lie95] Liedtke, Jochen: On Microkernel Construction. In: *15th ACM Symposium on Operating System Principles (SOSP)*. Copper Mountain Resort, CO, USA, Dezember 1995, S. 237–250

[MoK06] *The Month of Kernel Bugs (MoKB) archive*. `http://projects.info-pull.com/mokb/`, 2006

[Sna08] *Secure Computing*. `http://www.snapgear.org`, 2008

[SPHH06] Singaravelu, Lenin ; Pu, Claton ; Härtig, Hermann ; Helmuth, Christian: Reducing TCB Complexity for Security-Sensitive Applications: Three Case Studies. In: *Proceedings of the First European System Conference (EuroSys 1)*. Leuven, Belgien, April 2006

[TMLL06] Ta-Min, Richard ; Litty, Lionel ; Lie, David: Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*. Seattle, WA, USA, November 2006

[TPM09] *Trusted Computing Group: TPM*. `https://www.trustedcomputinggroup.org/groups/tpm/`, 2009

[WH08] Weinhold, Carsten ; Härtig, Hermann: VPFS: Building a Virtual Private File System with a Small Trusted Computing Base. In: *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. Glasgow, Schottland, UK, April 2008