µSINA – Eine mikrokernbasierte Systemarchitektur für sichere Systemkomponenten[1]

Christian Helmuth[2], Andreas Westfeld[2], Michael Sobirey[3]

Im Rahmen des Forschungsprojekts µSINA erfolgt die Entwicklung einer Betriebssystemarchitektur für Systeme mit sehr hohen Sicherheitsanforderungen. Diese Architektur weist eine überschaubare Komplexität auf und ermöglicht dadurch eine effiziente Evaluierung. Die Systemarchitektur von µSINA wird exemplarisch für ein Sicherheitsgateway auf Mikrokernbasis umgesetzt, dessen Komponenten in separaten Adressräumen ablaufen und dadurch voreinander geschützt sind. Diese können ausschließlich über einen Kernmechanismus miteinander kommunizieren. Auf diese Weise wird unkontrollierte Kommunikation unterbunden und die Auswirkung von Fehlern in einzelnen Komponenten eingeschränkt. Die so entkoppelten Komponenten lassen sich unabhängig voneinander evaluieren.

1 Einführung

In den vergangenen drei Jahren entstand, basierend auf einem Konzept des BSI im Rahmen eines Entwicklungsauftrages mit SINA (Sichere Inter-Netzwerk-Architektur) [BSI 2002] eine modulare, flexibel skalierbare Architektur für eine sichere Verarbeitung und Übertragung von Verschlusssachen (VS) sämtlicher nationaler und wichtiger internationaler Geheimhaltungsstufen. Die in SINA integrierten Sicherheitstechnologien umfassen u. a. IPSec (VPN), sowohl software- als auch hardwarebasierte Kryptographie, eine Public-Key-Infrastruktur, Smartcards, sicheres Löschen, IP-Paketfilter, ein kombiniertes Betriebssystem- und rechnerspezifisches Netzaudit sowie Intrusion Detection & Response. Im Jahr 2002 erreichten die ersten drei Basiskomponenten, der SINA Thin Client, die SINA Box – ein IPSec-Gateway [Kent Atkinson 1998] – und das dazugehörige SINA Management den finalen, spezifizierten Funktionsumfang.

Den beiden erstgenannten Komponenten liegt SINA-Linux zugrunde, eine extrem minimalisierte und funktional „gehärtete“ Systemplattform, deren Sicherheit aufwändig analysiert wurde. Während auf Prozessebene vorrangig extensiv minimalisiert wurde, konzentrierten sich auf Kernebene die Anstrengungen auf Absicherung bzw. Abgrenzung der unterschiedlichen Datenströme gegeneinander. Der Codeumfang der SINA Box beläuft sich beispielsweise auf (komprimierte) 5 MByte und ermöglicht damit auch das Booten von Flash-Speichermedien.

Ziel des Forschungsprojektes µSINA ist es, eine vereinfachte Variante der SINA Box auf eine, gegenüber dem bislang verwendeten monolithischen Betriebssystem nochmals signifikant schlankere und damit besser evaluierbare Systemplattform abzubil-

---

[1] Dieser Beitrag entstand während der Bearbeitung des vom Bundesministerium für Wirtschaft und Arbeit im Rahmen des Forschungsprogramms VERNET (http://www.vernet-info.de) geförderten Projektes „µSINA“.

[2] Technische Universität Dresden, Institut für Systemarchitektur

[3] secunet Security Networks AG

den, prototypisch umzusetzen und zu erproben. Außerdem wird eine noch differenziertere Separierung von Prozessen und Daten(strömen) angestrebt. Grundlage hierfür ist eine mikrokernbasierte Systemplattform. Die o. g. Variante der SINA Box wird schrittweise auf diese Systemplattform portiert.

2 Mikrokerntechnologie und Sicherheit

2.1 Aktueller Stand der Mikrokerntechnologie

Während im *Kern* eines monolithischen Betriebssystems *alle* grundlegenden Systemdienste integriert sind, wird beim mikrokernbasierten Ansatz insbesondere dieser Teil minimiert. Bei letzterem werden alle Dienste in Mikrokernserver ausgelagert, die in unabhängigen Adressräumen ablaufen. Die Mikrokernserver nutzen die gleichen Kernmechanismen wie Nutzerprogramme. Dadurch können Programmierfehler oder fehlerhafte Konfigurationen von Systemdiensten und Nutzerprogrammen gleichermaßen nicht manipulierend auf den Kern und andere Systemkomponenten übergreifen. Tritt in einer Komponente ein Problem auf, so kann diese umkonfiguriert und neu gestartet werden, ohne dass ein Neustart des gesamten Betriebssystems erforderlich ist. Das erweist sich als hilfreich in Anwendungsbereichen, die eine hohe Verfügbarkeit erfordern und erhöht sowohl die Sicherheit als auch die Robustheit des Gesamtsystems. Dies wird durch den sehr schlanken Kern im privilegierten Prozessormodus und Betriebssystemdienste in separaten Adressräumen erreicht.

Bisherige Mikrokerne (z. B. [Bridges 1998]) erreichten nur geringe Akzeptanz, da sich die Leistung eines Großteils der bekannten Implementierungen als unzureichend erwies. Dies beeinträchtigt vor allem die Flexibilität, da wichtige Kernmechanismen aufgrund schlechter Performance nicht genutzt werden. Eine gängige Praxis für Mikrokerne der ersten Generation war es, entgegen dem minimalisierenden Ansatz, Dienste in den Kern zu reintegrieren, obwohl der Mikrokernansatz nicht die Ursache des Leistungsproblem ist [Liedtke 1995].

Ergebnis eines neuen Vorstoßes in Richtung „echter“ Mikrokerne sind die Vertreter der zweiten Generation. Diese erreichen aufgrund sorgfältiger Analyse der kritischen Pfade innerhalb des Kerns eine weitaus höhere Leistung und beschränken sich dabei auf ein Minimum an Kernmechanismen. Der Mikrokern *Fiasco* [Hohmuth 1998] der TU Dresden ist ein Kern der zweiten Generation und implementiert die L4-Schnittstelle [L4]. Er zeichnet sich durch die Programmierung in einer Hochsprache (C++) und sehr gute IPC[4]-Performance aus.

2.2 Vergleich zu Multilevel Security und Domain Type Enforcement

Multilevel Security (MLS) oder das Bell-LaPadula-Modell [Bell LaPadula 1973] organisiert Daten und Anwendungen, welche verschiedenen Sicherheitsstufen angehören, auf Sicherheits- oder Gefährdungsebenen. Der Austausch von Daten ist festen Re-

[4] IPC = Interprozesskommunikation oder *Message Passing*

geln unterworfen (Mandatory Access Control), die nicht vorsätzlich oder versehentlich umgangen werden können:

- Anwendungen können auf all jene Objekte lesend und schreibend zugreifen, die sich auf der gleichen Sicherheitsstufe befinden.
- Keine Anwendung kann Daten einer höheren Sicherheitsstufe lesen – *no read up (NRU).*
- Keine Anwendung kann Daten in eine niedrigere Sicherheitsstufe schreiben – *no write down (NWD).*

Da alle Aktivitäten diesen Regeln unterworfen sind, ist es auch trojanischen Pferden nicht möglich, Daten in niedrigere Sicherheitsstufen zu transferieren. In der Praxis sind für MLS-Systeme vor allem verdeckte Kanäle problematisch, da verschiedene Sicherheitsstufen auf gemeinsame Ressourcen (z. B. Dateisystem, Geräte) zugreifen und somit zum Transfer von Informationen nutzen können.

Seit dem Ende der 80er Jahre fanden MLS-Systeme vor allem im militärischen Bereich praktische Umsetzungen. Ein Beispiel ist SCOMP, eine um MLS erweiterte Version des Betriebssystems MULTICS. Gegenwärtig wird MLS vor allem in modifizierten Versionen von Unix, wie z. B. V/MLS von AT&T oder SGIs Trusted IRIX angewendet.

Die von MLS verlangte Einteilung in Sicherheitsstufen erwies sich als unflexibel und schwierig im praktischen Einsatz [Anderson 2001]. Neben anderen Weiterentwicklungen ist besonders das Domain-Type-Enforcement interessant, das neben der Vertraulichkeit auch die Integrität der Daten betrachtet. Prozesse werden hierbei *Domänen* und Objekte (u. a. Daten) *Typen* zugeordnet. Eine Zugriffsmatrix – die *Domain Definition Table (DDT)* – regelt die Zugriffe von Domänen auf Typen. So wird jeder Aktivität explizit der Zugriff auf unbedingt benötige Betriebsmittel gestattet (Prinzip der geringstmöglichen Privilegierung). Der gemeinsame Zugriff von Prozessen auf globale Betriebsmittel kann hiermit auf ein Minimum reduziert werden. Dadurch werden die Möglichkeiten für verdeckte Kanäle drastisch eingeschränkt.

Das Domain-Type-Enforcement wird gegenwärtig kommerziell in Firewalls, wie z. B. Sidewinder, eingesetzt [Smith 1996] [Secure Computing 2000]. Als Basis dient hier LOCK (Logical Coprocessing Kernel), welcher seit 1987 entwickelt wird [Smith 2001]. Die Durchsetzung der Zugriffsrechte wird durch den darauf aufbauenden Referenzmonitor SIDEARM (System Independent Domain Enforcing Assured Reference Monitor) als Systemkomponente erzwungen. Der Entwurf von SIDEARM kombiniert Software und spezielle Hardware.

µSINA setzt das Prinzip der geringstmöglichen Privilegierung durch den Einsatz eines Mikrokerns auf Standard-PC-Hardware um. Die Kommunikation zwischen einzelnen Systemkomponenten findet per Interprozesskommunikation (IPC) und somit durch den Mikrokern kontrolliert statt.

3 Architektureller Ansatz für µSINA

Die Systemarchitektur von µSINA setzt eine sichere Systemplattform als Basis für anwendungsspezifische Dienste voraus. Hierbei wird die Hardware des Systems als vertrauenswürdig angesehen, d. h. alle Geräte verhalten sich entsprechend ihrer Spezifikation und beinhalten keine darüber hinausgehende (versteckte) Funktionalität. Diese Annahme gilt für Standardgeräte wie CPU und Speicher genauso wie für Erweiterungskarten, z. B. Netzwerkadapter.

Weitere grundlegende Anforderungen an die Systemplattform für µSINA sind:

- Die Komplexität ist ausreichend überschaubar für sorgfältige Evaluation.
- Eine Separierung der Dienste ist möglich, so dass unkontrollierte Kommunikation und die Auswirkungen von Fehlern einzelner Komponenten auf das Gesamtsystem unterbunden bzw. eingeschränkt werden.
- Die Privilegien der Systemkomponenten können auf das nötige Minimum reduziert werden, was als *Prinzip der geringsten Privilegierungsstufe* bezeichnet wird. Damit stellt die Systemplattform sicher, dass der Einflussbereich der Komponente von vornherein begrenzt ist.
- Die Systemressourcen werden verwaltet und genau *eine* Vergabe- und Sicherheitspolitik verlässlich durchgesetzt.

Um diese Anforderungen zu erfüllen, wird in µSINA ein mikrokernbasierter (und kein monolithischer) Ansatz für die Architektur des Betriebssystems verfolgt. Somit werden alle Komponenten des Systems durch die vom Mikrokern implementierten Adressraumgrenzen separiert. Adressräume bilden auf Hardwareebene die Seiten des virtuellen Adressraums der Komponente auf physische Seiten des Hauptspeichers ab. Diese Abbildung wird durch die TLB-Hardware (Translation Lookaside Buffer) der CPU und Seitentabellen implementiert, deren Modifikation dem Mikrokern vorbehalten ist.

Die Änderung der Adressabbildung kann nur mit Hilfe des Mikrokerns durchgeführt werden. Somit sind alle Komponenten des Systems durch Adressraumgrenzen untereinander geschützt und können diese Grenzen ausschließlich mit Hilfe der vom Kern angebotenen und kontrollierten Mechanismen überwinden.

Der Nachrichtenmechanismus des Mikrokerns wird mittels IPC oder *Message Passing* realisiert. Der Austausch von Botschaften findet nur dann statt, wenn *beide* Kommunikationspartner diesem zustimmen. Außerdem bietet der Mikrokern einen Mechanismus zur Kontrolle des Message Passings (z. B. [Liedtke 1992]). Mit Hilfe dieses Mechanismus ist es möglich, die Kommunikation auf bestimmte Partner einzuschränken. Die so kontrollierten Komponenten erhalten folglich nur die zur Erfüllung ihrer spezifischen Aufgabe nötigen kommunikationsrelevanten Privilegien.

Eine Laufzeitumgebung bestehend aus Mikrokernservern verwaltet die Systemressourcen mit Hilfe der Mechanismen des Kerns (siehe Abb. 1). Die Ressourcen sind im Einzelnen Hauptspeicher, Adressräume und Threads sowie Ein-/Ausgabe-Ressourcen

(I/O-Ports, Memory Mapped I/O-Bereiche, Interrupts). Die *Ressourcenmanager* setzen auf Grundlage der vom Mikrokern angebotenen Mechanismen *eine* systemspezifische Ressourcen-Vergabepolitik durch und sind somit ein Teil der Systemplattform und als vertrauenswürdig einzustufen. Zur Laufzeitumgebung zählt auch ein sicherer Namensdienst, der symbolische Dienstnamen auf eindeutige Thread-Identifikatoren abbildet.

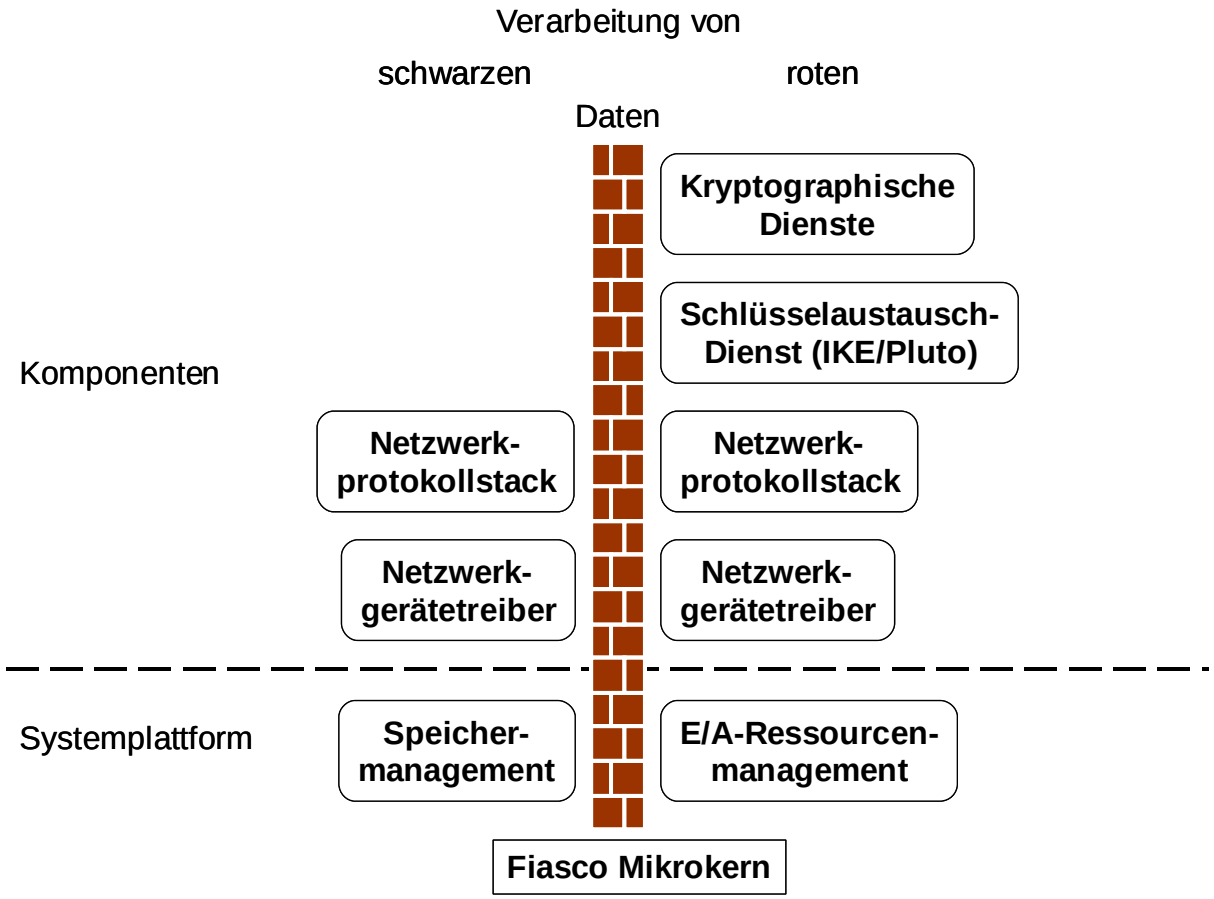

Abbildung 1: Die Systemarchitektur von µSINA

Auf der Grundlage dieser Plattform werden nun die weiterführenden Dienste für µSINA in separaten Komponenten umgesetzt.

4 Exemplarische Umsetzung

Nachdem die Entwicklung der Systemplattform abgeschlossen ist, besteht die zweite Aufgabe in µSINA in der schrittweisen Portierung einer Variante der SINA Box auf diese Plattform. Dieses Sicherheitsgateway trennt physisch zwei Netze, indem es für jedes der Netze einen Netzwerkadapter enthält und die Netzwerkpakete mit Hilfe der Systemsoftware von einem Adapter zum anderen weiterleitet. Die beiden Netze werden als *roter* bzw. *schwarzer Bereich* aufgrund der Sichtbarkeit der sensitiven Daten bezeichnet. So sind die sensitiven Daten im *roten Bereich* nicht gegen Zugriffe oder Veränderungen geschützt, während diese im *schwarzen Bereich* zur Einhaltung von

Vertraulichkeit und Integrität verschlüsselt („geschwärzt“) sind und einen Authentifikationscode (MAC) tragen.

Innerhalb des Sicherheitsgateways existieren verschiedene Komponenten, die Daten aus dem roten und schwarzen Bereich verarbeiten. Die Komponenten des Systems laufen in separaten Adressräumen ab, was unkontrollierte gegenseitige Beeinflussung ausschließt. Es muss nun entschieden werden, welche Komponenten des Systems evaluiert werden müssen, d. h. vertrauenswürdig sind.

Zur Einhaltung der Schutzziele Vertraulichkeit und Integrität wird gefordert, dass Daten, welche vom roten in den schwarzen Bereich gelangen, kryptographisch gegen Einsicht und Manipulation durch potentielle Angreifer geschützt sind. Das bedeutet, dass Komponenten, welche Daten beider Bereiche verarbeiten, vertrauenswürdig sein müssen, also Gegenstand der Evaluierung sind.

Wenn eine Komponente nur schwarze Daten verarbeitet und keine Möglichkeit hat, direkten Einfluss auf andere Systemkomponenten zu nehmen, kann diese von der Evaluation ausgeschlossen werden. Unsere Architektur setzt dies mittels Adressräumen und IPC-Redirection durch.

Die Verarbeitung von roten Daten ist sicherheitskritisch und muss vertrauenswürdig durchgeführt werden. Eine Evaluation aller Komponenten, welche rote Daten verarbeiten, würde diese Anforderung erfüllen. Da aber möglichst wenige Komponenten evaluiert werden sollen und der Anteil der sicherheitskritischen innerhalb des Gateways groß ist, werden diese nochmals genauer betrachtet.

In [Härtig 2002] wird das *Tunneling* als eine Technik zur Minimierung der Anzahl der Komponenten in der Systemplattform beschrieben. Hier wird vorgeschlagen, z. B. Dateisysteme nicht als Bestandteile einer sicheren Systemplattform zu betrachten und von einer Evaluation auszuschließen, wenn durchgesetzt wird, dass diese nur verschlüsselte Daten verarbeiten. Das Tunneling wird als Technik definiert, die es erlaubt, „Software zu verwenden, welche eine verlangte Eigenschaft nicht erfüllt, und diese Eigenschaft in einer zusätzlichen Schicht durchzusetzen“.

Eine Möglichkeit diesen Ansatz fortzuführen ist, Komponenten vollständig vom restlichen System zu isolieren (Kapselung), so dass keine ungewollte Kommunikation möglich ist. Allgemein muss eine solche Komponente nur minimale Handlungsfreiheit erhalten. Diese Forderung bezeichnet man als Prinzip der geringsten Privilegierung. Auf diese Weise ist eine kryptographische Behandlung der verarbeiteten Daten nicht nötig, da Absender und Empfänger festgelegt sind.

Die Systemarchitektur von µSINA bietet alle nötigen Mittel, um diese Anforderungen zu erfüllen: Adressräume und kontrollierte IPC. Es ist somit möglich, Inseln zu schaffen, auf denen nicht vertrauenswürdige Komponenten rote Daten verarbeiten.

Ein Problem sind verdeckte Kanäle *(Covert Channels)* zwischen Komponenten des Systems, welche mit unserer Technik nicht ausgeschlossen werden können. Eine Voraussetzung für verdeckte Kanäle sind gemeinsam benutzte Ressourcen, die man, da die Komponenten auf einem System ablaufen, nicht vollständig vermeiden kann.

In den folgenden beiden Abschnitten gehen wir genauer auf zwei integrale Bestandteile des µSINA-VPN-Gateways ein – den Netzwerkstack und die Netzwerkgerätetreiber.

4.1 Netzwerk-Protokollstack

Die Implementierung der Netzwerkprotokolle wird als Stack realisiert, so dass Protokolle der Transportschicht (TCP, UDP) auf der Netzwerkschicht (IP, IPSec) aufbauen und Netzwerkpakete durch diese und weitere Schichten „wandern“. Da IPSec ein wichtiger Bestandteil der Netzwerksoftware ist, werden schwarze und rote Daten verarbeitet. Die Pakete werden auf dem Weg durch den Stack kryptographisch zum Schutz von Vertraulichkeit oder Integrität behandelt. Deshalb müssen sicherheitskritische Bestandteile evaluiert werden.

Die Komplexität aktueller Implementierungen ist sehr hoch. So umfassen z. B. die IP-spezifischen Module innerhalb des Linuxkerns (Version 2.4.20) etwa 80 000 Zeilen C. Deshalb und auf Grund der Tatsache, dass von zukünftigen Entwicklungen leichter profitiert werden kann, ist die Wiederverwendung einer bestehenden Umsetzung einer Neuimplementierung vorzuziehen.

Um den zu evaluierenden Quellcode so gering wie möglich zu halten, bieten sich unter diesen Voraussetzungen zwei Möglichkeiten:

Zergliederung der bestehenden Software in mehrere sicherheitskritische und -unkritische Komponenten

Für die Zergliederung müssen die Komponenten im bestehenden System identifiziert und anschließend gekapselt werden. Als Beispiel soll das in Abbildung 2 dargestellte Szenario dienen:

1. Komponente: IPSec-Policy
   Die Policy entscheidet für jedes Paket, ob es verarbeitet wird. (Falls Verschlüsselung für eintreffende Pakete erwartet aber nicht angewendet wurde, werden diese verworfen.)
2. Komponente: Routing
   Das Routing bestimmt die weitere Verfahrensweise für das Paket – lokale Verarbeitung oder Weiterleitung – und modifiziert die Paketheader entsprechend.
3. Komponente: IPSec-Konzelation
   In dieser Komponente werden die Pakete kryptographisch behandelt.

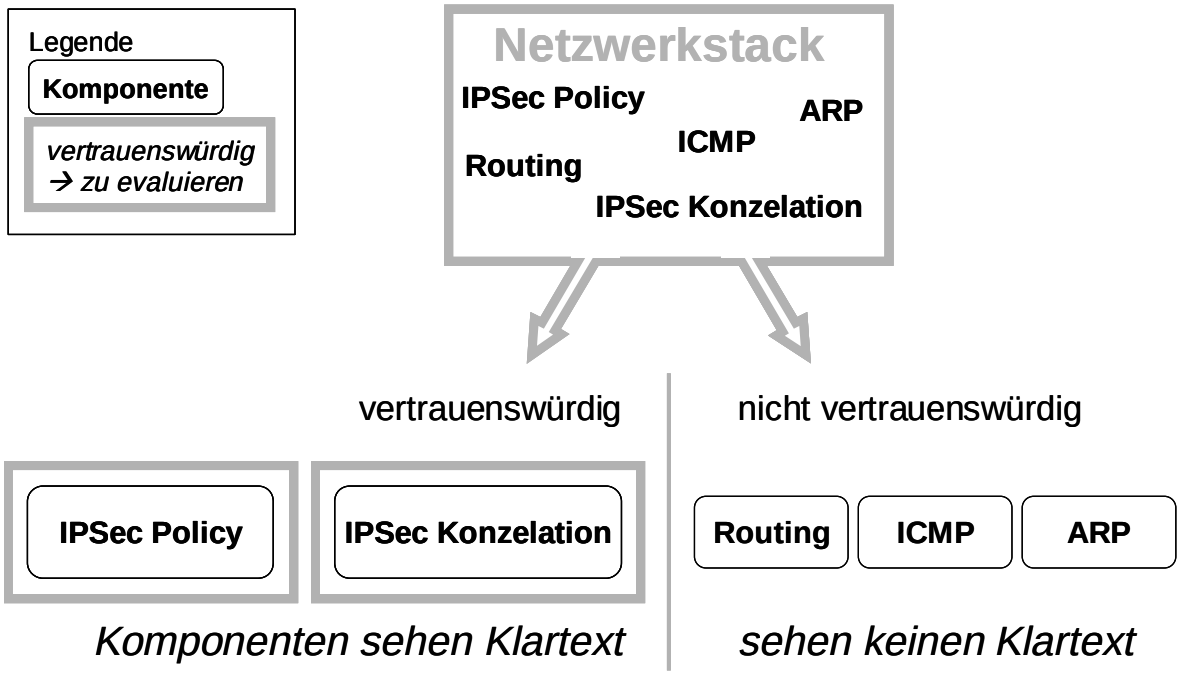

Abbildung 2: Zergliederung des Netzwerkstacks

Von diesen Komponenten ist das Routing für die Vertraulichkeit und Integrität der Daten unkritisch, da hier nur die Header (Verkehrsdaten) des Pakets verarbeitet werden. Die in Abbildung 2 dargestellte Zergliederung schließt das Routing von den zu evaluierenden Komponenten aus. Genauso können nun weitere Komponenten betrachtet werden.

Diese Vorgehensweise differenziert die Netzwerksoftware in mehrere kleine Module und erlaubt die spezifische Verringerung des zu evaluierenden Quelltextes. Es ist anzumerken, dass hier eventuell komplexere Schnittstellen eingeführt werden als Bestandteil der Originalimplementierung sind. Außerdem zieht eine starke Zergliederung die Einführung neuer Schnittstellen innerhalb bestehender Module nach sich, so dass die Wiederverwendbarkeit der neuen Komponenten bei Aktualisierungen der zugrunde liegenden Originalimplementierung nicht gewährleistet sein muss. Modulinterne Änderungen des Originals sind sehr wahrscheinlich, auch wenn die Schnittstelle unverändert bleibt.

Verwendung von zwei Netzwerkstacks und einer Brückenkomponente

Eine Möglichkeit, fest definierte Schnittstellen beizubehalten, ist, die Netzwerkprotokollfunktionen zu duplizieren und zwei komplette Stacks in einem System zu verwenden. Kommunikation zwischen diesen kann nur über eine „Brücke“ stattfinden, die eine IPSec-konforme Ver- und Entschlüsselung der Netzwerkpakete durchsetzt.

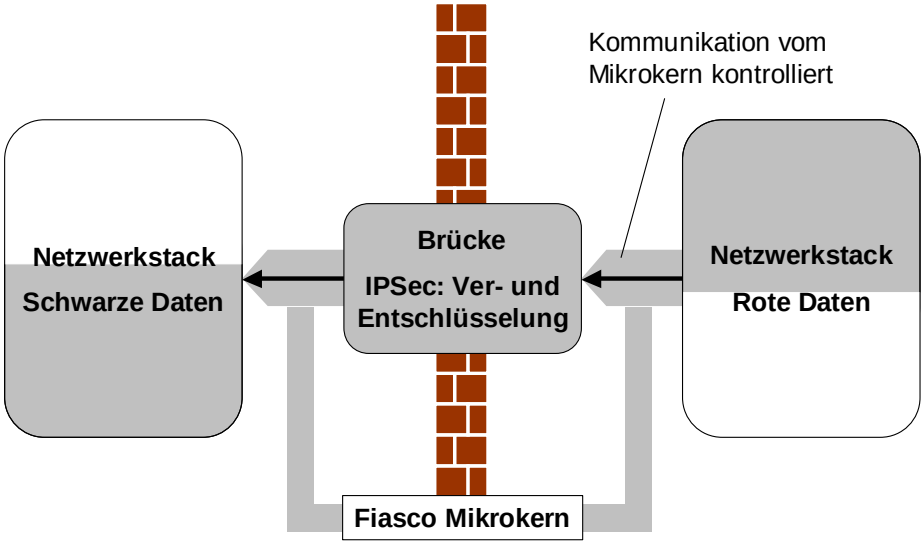

Abbildung 3: Realisierung der Netzwerkprotokolle mit Brückenkomponente

Bei diesem Ansatz ist gewährleistet, dass jeder Stack nur Daten eines Bereiches – rot oder schwarz – verarbeitet. Gegenstand einer Evaluierung ist hierbei nur die Brückenkomponente, die teilweise von bestehenden IPSec-Implementierungen abgeleitet sein wird. Die Netzwerkstacks müssen so angepasst werden, dass die Daten (Netzwerkpakete) auch von zwischengelagerten Ebenen nach außen gegeben werden. Die Brücke verbindet die beiden Teil-Stacks zu einem kompletten Netzwerk-Protokollstack (siehe Abbildung 3).

Es stellt sich nun die Frage, wie angreifbar diese Lösung ist. Da beide Netzwerkstacks nicht evaluiert werden, sind diese durch potenziell eingeschleuste Trojanische Pferde gefährdet. Sie allein bieten somit keine gegenüber dem Originalsystem gesteigerte Sicherheit. Die Brückenkomponente ist Gegenstand einer Evaluation und somit gegen bekannte Angriffe immun. Weiterhin wird die gesamte Kommunikation zwischen den Stackkomponenten kryptographisch behandelt und potenzielle Trojanische Pferde haben deshalb keine Möglichkeit für den offenen Austausch von Informationen. Durch die Brücke wird auch bei kompromittierten Protokollimplementierungen durchgesetzt, dass keine sensitiven Daten in die unteren Schichten des Netzwerkstacks gelangen und Vertraulichkeit sowie Integrität geschützt sind.

Verdeckte Kanäle wurden von uns nicht näher betrachtet, können aber keinesfalls ausgeschlossen werden. Innerhalb der Brückenkomponente können bekannte Techniken eingesetzt werden, z. B. Pump [Kang Moskovitz 1993], um gemeinsam benutzte Betriebsmittel der Netzwerkstacks zu entkoppeln und die Kommunikation als Trägermedium auszuschließen.

4.2 Gerätetreiber

Die Ansteuerung von Ein-/Ausgabe-Geräten trägt aufgrund der Vielfalt der verfügbaren Chipsätze und Hersteller beträchtlich zur Komplexität aktueller Betriebssysteme bei. Daher würde eine Verlagerung der Gerätetreiber aus der Systemplattform in eine

möglicherweise nicht vertrauenswürdige Komponente einen großen Beitrag zur Senkung der Komplexität der Systemsoftware leisten.

In diesem Zusammenhang muss die Funktionsweise der Ein-/Ausgabe-Geräte und der Ansteuerungssoftware auf der (PC-) Zielplattform genauer betrachtet werden. Hier wird häufig zur Entlastung der CPU das Kopieren der Daten zwischen Hauptspeicher und Gerät von externen Controllern – Direct Memory Access (DMA) – durchgeführt, die bei PCI-Adapterkarten integriert sind (siehe Abbildung 4).

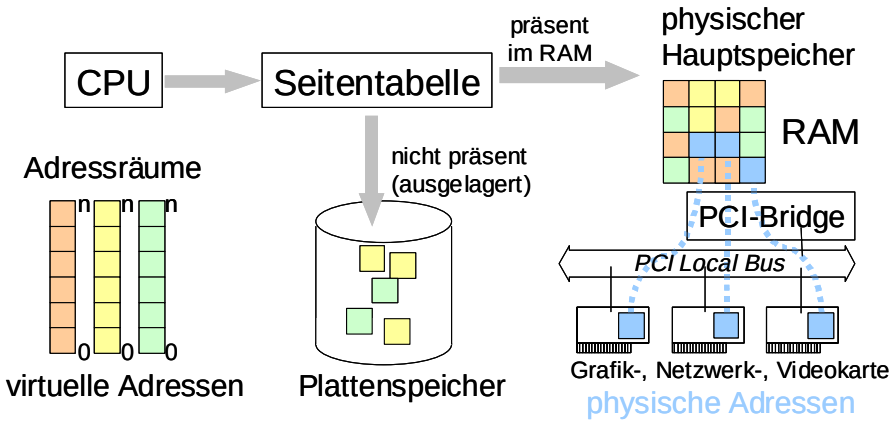

Abbildung 4: Busmaster-DMA-Zugriff auf physische Adressen

Diese Vorgehensweise verursacht aber ein Sicherheitsrisiko, das aus der Verwendung der beschriebenen *Busmaster-DMA-Controller* resultiert. Da diese nicht an die Adressabbildung der CPU gebunden und frei programmierbar sind, ist es mit Hilfe von DMA Komponenten des Systems, z. B. Gerätetreibern, möglich, über den Umweg der Geräteprogrammierung die vom Mikrokern durchgesetzten Adressraumgrenzen zu durchbrechen. Eine fehlerhaft programmierte Netzwerkkarte könnte z. B. auf den physischen Speicher zugreifen und dabei Teile des Mikrokerns überschreiben, ohne dass die CPU dabei eine Schutzverletzung feststellt.

Dieses Problem wird auch in [Härtig 2002] diskutiert und es werden zwei Lösungsansätze vorgeschlagen, um Gerätetreiber außerhalb der Systemplattform zu implementieren:

1. DMA für nicht vertrauenswürdige Komponenten wird verboten. Dies ist möglich indem die Geräte virtualisiert werden bzw. die DMA-Programmierung in eine vertrauenswürdige Komponente ausgelagert wird, die zwischen Gerät und Treiber vermittelt. Probleme bereitet an dieser Lösung, dass zum Einen Virtualisierung nicht für alle Geräte vollständig möglich ist (z. B. Timing-Probleme) und zum Anderen die Verwendung eines Vermittlers die aufwendige Analyse der gerätespezifischen Schnittstellen verlangt.

   Es ist anzunehmen, dass ein Großteil der Gerätetreibersoftware evaluiert werden muss, um die in Hinblick auf Performance nicht unkritischen DMA-Transfers zu erlauben.

2. Die DMA-Zugriffe werden mit Hilfe geeigneter Hardware auf Bereiche des physischen Speichers beschränkt. Somit kann die Systemplattform Speicher für Komponenten und Speicher für Ein-/Ausgabe voneinander getrennt verwalten. Dieser Ansatz wurde schon früher verfolgt [Fraim 1983], hat aber auch Nachteile. So ist die benötigte Hardware noch nicht Bestandteil von Standardsystemen. Es existieren momentan wenige Speziallösungen in Form von PCI-to-PCI-Bridges oder für andere Architekturen als PC-Systeme (Alpha [DEC 1996], SUN SBus DVMA [SUN 1997]).

Da der zweite Ansatz für µSINA vielversprechender ist, soll er hier noch etwas genauer analysiert werden.

Die genannte Lösung erlaubt eine Partitionierung des physischen Speichers in zwei Bereiche – DMA zugelassen und DMA verboten. Darüber hinaus ist aber eine Unterscheidung der Zugriffe verschiedener E/A-Geräte für die Sicherheit notwendig, da es sonst via Gerät A möglich ist, die Daten, welche Gerät B mit dem DMA-Speicherbereich austauscht, zu lesen oder zu modifizieren. Es gäbe somit einen Weg, die Adressraumgrenzen zu überwinden. Ohne Unterscheidung der Geräte wäre lediglich der Einflussbereich auf Ein-/Ausgabe-Puffer eingeschränkt.

Um diesen Mangel zu beheben, ist eine feingranulare Abbildung von Geräten auf Speicherbereiche (im Gegensatz zur Abbildung gesamte Ein-/Ausgabe auf Speicher) notwendig. Um diese Forderung prototypisch umzusetzen, müsste z. B. ein Gerät zwischen dem eigentlichen Ein-/Ausgabe-Gerät und dem PCI-Bus installiert werden, welches während des Adresszyklus nur bestimmte Leitungen des AD-Busses (multiplexed address and data bus) frei verwendbar lässt und die restlichen auf definierte Werte schaltet. Dies ermöglicht eine hardwareseitige Festlegung der Speicherpartitionierung, welche das System in seine Speichervergabestrategie einbeziehen kann.

Eleganter und dynamisch konfigurierbar ist eine Implementierung der Speicheraufteilung im Controller zwischen Geräte- und Speicherbus (Host-Bridge). Die Systemsoftware erhält damit die vollständige Kontrolle über den Hauptspeicher und kann durch exklusive Programmierung des Controllers eine geeignete Aufteilung dessen für Komponenten und Geräte durchsetzen.

In einem System, das eine hinreichende Lösung der Busmaster-DMA-Problematik implementiert, sind auch Gerätetreiber fest an die von der Systemplattform vorgegebenen Adressräume gebunden – auf CPU- und auf Geräteseite. Die Treiber können dann in der Systemarchitektur von µSINA außerhalb der Systemplattform liegen und müssen nicht zwingend evaluiert werden.

5 Ausblick

Im vorliegenden Beitrag wurde eine Architektur für Betriebssysteme (µSINA) vorgestellt, die sowohl sehr hohe Sicherheit gewährleistet als auch effizient evaluierbar ist. Die Verwendung einer kleinen, mikrokernbasierten Plattform zur Erreichung dieser Leistungsmerkmale ist vielversprechend, da die Differenzierung der Systemsoftware

in eigenständige Komponenten die Komplexität überschaubar hält und die Komponenten voreinander schützt.

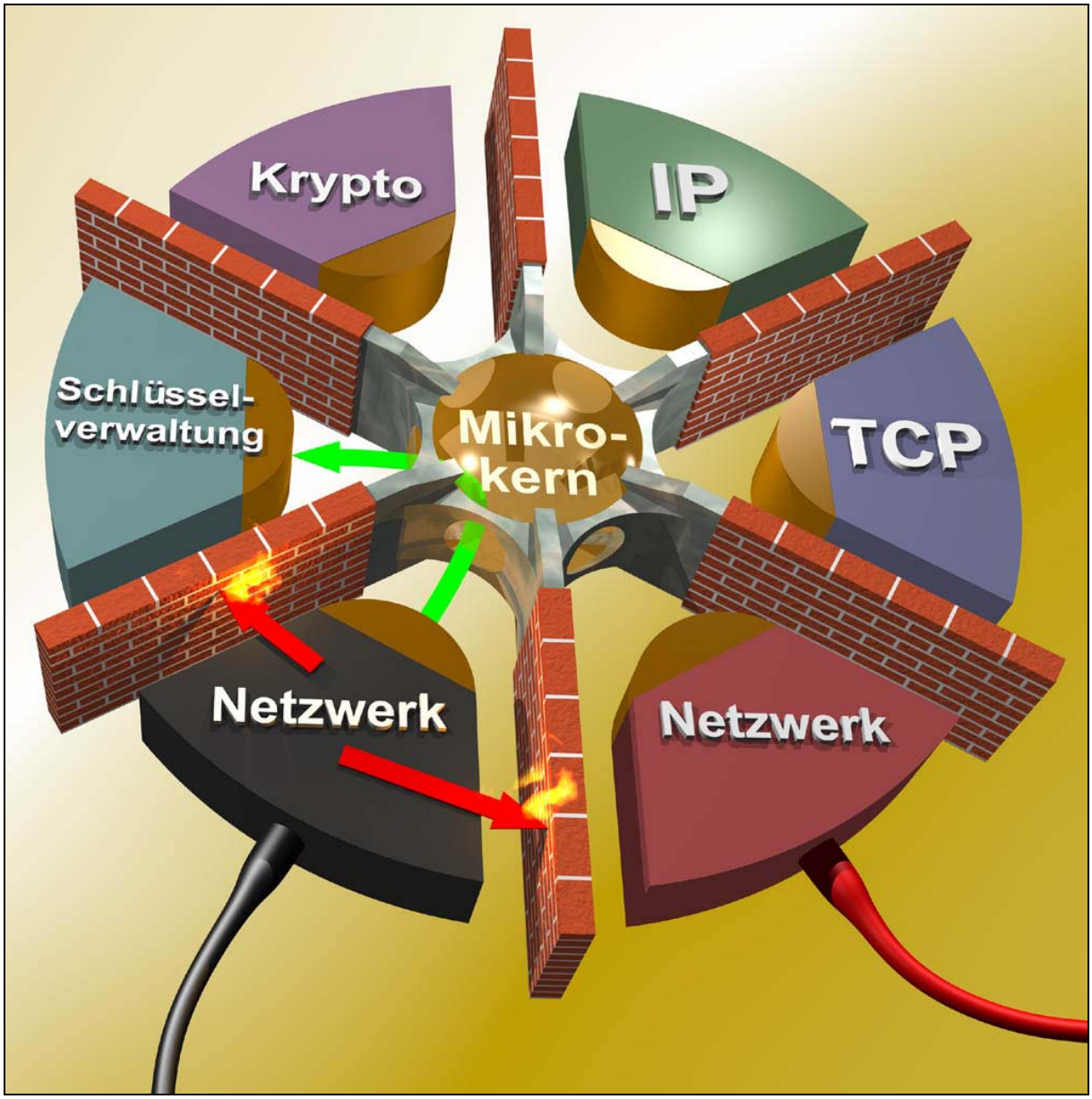

Abbildung 5: Mittels Mikrokern kontrollierte Kommunikation

Abbildung 5 veranschaulicht, dass unsere mikrokernbasierte Sicherheitsarchitektur auch für andere Einsatzfelder als Sicherheitsgateways geeignet ist und die Komponenten entsprechend austauschbar sind. Weitere potenzielle Anwendungsbereiche sind beispielsweise

- Web-Server,
- Router,
- Firewalls,

- Intrusion Detection & Response Systeme und
- mobile Geräte.

Darüber hinaus wurde festgestellt, dass es für µSINA (und andere mikrokernbasierte Systeme) dienlich wäre, die Behandlung von Busmaster-DMA in PC-Systemen zu ändern, so dass die Systemplattform die Hauptspeicherzugriffe vollständig kontrolliert. Komplexe Gerätetreiber müssten dann nicht evaluiert werden bzw. Bestandteil der Plattform sein.

Quellenverzeichnis

[Anderson 2001] Anderson, R., Security Engineering, John Wiley & Sons, New York, Chichester, 2001.

[Bell LaPadula 1973] Bell, D. E., LaPadula, L.: *Secure Computer Systems*, MITRE Corporation, 1973.

[Bridges 1998] Bridges, P.: Microkernel-based and Similar Operating Systems, URL: http://www.cs.arizona.edu/people/bridges/os/microkernel.html

[BSI 2002] Bundesamt für Sicherheit in der Informationstechnik: *Sichere Inter-Netzwerk-Architektur*, 2002.
URL: http://www.bsi.bund.de/fachthem/sina/index.htm.

[DEC 1996] Digital Equipment Corporation: *Digital Semi-conductor 21172 Technical Reference Manua*l, 1996. (Abschnitt 6.4: PCI to Physical Memory Addressing).

[Fraim 1983] Fraim, L.J.: *Scomp: A Solution to the Multilevel Security Problem*, Computer, 16(7): 26–34, 1983.

[Härtig 2002] Härtig, H.: *Security Architectures Revisited*, 10th ACM SIGOPS European Workshop, September 2002.
URL: http://os.inf.tu-dresden.de/papers_ps/secarch.pdf.

[Hohmuth 1998] Hohmuth, M.: *The Fiasco Microkernel: Requirements Definitio*n. Technischer Bericht TUD-FI98-12, TU Dresden, Dezember 1998.
URL: http://os.inf.tu-dresden.de/fiasco.

[Kang Moskovitz 1993] Kang, M. H., Moskovitz, I. S.: *A pump for rapid, reliable, secure communication*, 1st ACM Conference on Computer and Communication Security, 1993, 119-129.

[Kent Atkinson 1998] Kent, S., Atkinson. R.: *Security Architecture for the Internet Protocol*. RFC 2401, Network Working Group, 1998.

[L4] L4 Developer's Bibliography: http://os.inf.tu-dresden.de/L4/bib.html

[Liedtke 1992] Liedtke, J.: *Clans & Chiefs*, 12. GI/ITG-Fachtagung Architektur von Rechnersystemen, März 1992. S. 294-305

[Liedtke 1995] Liedtke, J.: *On µ-Kernel Construction*, 15th ACM Symposium on Operating System Principles (SOSP), Dezember 1995.

[Secure Computing 2000] Secure Computing Corporation: *Type Enforcement Technology for Access Gateways and VPNs*, 2000.
URL: http://www.securecomputing.com/pdf/type_enforcement_wp.pdf

[Smith 2001] Smith, R. E.: *Cost Profile of a Highly Assured, Secure Operating System*, März 2001. URL: http://www.smat.us/crypto/docs/Lock-eff-acm.pdf.

[Smith 1996] Smith, R. E.: *Sidewinder: Defense in Depth using Type Enforcement*, International Journal of System Management, 1996.
URL: http://www.securecomputing.com/pdf/sdw-te.PDF

[SUN 1997] SUN Microsystems, INC.: *U2S User's Manual*, 1997. (Kapitel 13: SBus IOMMU).