KERNKONZEPT

Abschlussbericht Projekt VerseCloud

Kernkonzept GmbH

11. Dezember 2024

- ZE: Kernkonzept GmbH
- Förderkennzeichen: 16KIS1357K
- Vorhabenbezeichnung: Microhypervisor für sicherheitskritische Cloudanwendungen mit formal verifiziertem Design
- Laufzeit des Vorhabens: 01.04.2021 - 30.09.2024

Inhaltsverzeichnis

1 Ursprüngliche Aufgabenstellung, sowie wissenschaftlicher und technischer Stand an den angeknüpft wurde

Kann eine Cloud hohen Sicherheitsanforderungen entsprechen und zugleich zertifiziert und zugelassen sein? Existierende Lösungen basieren auf monolithischen Systemarchitekturen. Dort läuft ein Großteil der Betriebssystemfunktionalität mit höchsten Privilegien. Die Trusted Computing Base (TCB), also der Code der korrekt sein muss, umfasst mehrere Millionen Zeilen – viel zu groß, um auditiert werden zu können. Mikrokernsysteme werden dagegen bereits in bis GEHEIM zugelassenen Produkten für Behörden eingesetzt. Kann auf Basis eines Mikrokernsystems eine effiziente und flexible Virtualisierungslösung implementiert werden?

Falls ja, kann durch formale Methoden die Konfidenz in die Implementierung erhöht werden? Vorarbeiten im Bereich formale Methoden haben bisher nur den Mikrokern selbst betrachtet. Reale Anwendungen benötigen aber wesentlich mehr Komponenten. Sind formale Methoden auf einen Cloudhypervisor anwendbar?

Gerätetreiber sind für Virtualisierung von hoher Bedeutung. Kann ihre Korrektheit mit Hilfe von formalen Methoden nachgewiesen werden? Falls ja, können sie zum Teil synthetisiert werden?

Moderne Cloudhypervisoren benutzen Snapshots: die Sicherung des kompletten Zustands eines Gastsystems. Dies ermöglicht Zurücksetzen (Rollback) oder Umziehen (Migration) virtueller Maschinen. Hohe Komplexität und geringe Komponentenseparation monolithischer Ansätze bieten auch hier größere Angriffsoberfläche als Mikrokernsysteme. Sie können daher kaum Garantien zur Integrität erstellter Snapshots machen. Das wäre aber nötig für vertrauenswürdige Rollbacks und Migrationen. Sind Mikrokerne mit performantem Snapshot-Mechanismus und zugesicherter Integriät möglich?

Die Leitfrage für das Projekt Versecloud ist:

Wie können Mikrokerne zu einem performanten, flexiblen und vertrauenswürdigen Hypervisor mit sehr hoher Konfidenz in die Korrektheit seiner Implementierung weiterentwickelt und um integritätsgeschützte Snapshotting-Werkzeuge erweitert werden?

2 Ablauf des Vorhabens

- Projektstart: 01.04.2021
- Projektende: 30.09.2024
- Oktober 2021: Uni Potsdam steigt in das Projekt ein
- März 2022: Mario Egger stößt als studentische Hilfskraft von Mario Frank zum Projekt
- März 2022: TUM verlässt das Projekt; Kernkonzept übernimmt in der Folge die Arbeitspakete der TUM

Das erste Projektjahr war geprägt von den Einschränkungen durch die Corona-Pandemie. Darum fand unser erstes Konsortialtreffen erst im Oktober 2021 statt.

Wir haben eine regelmäßige Videokonferenz eingerichtet, in der wir uns im Rhythmus von 2 Wochen trafen um unser Vorgehen abzusprechen. Zusätzlich haben wir eine Mailingliste eingerichtet sowie ein Projektwiki angelegt.

Nach einem Projektjahr ist unser Konsortialpartner Technische Universität München (TUM) aus dem Projekt ausgestiegen. Um das Projektziel dennoch zu erreichen hat Kernkonzept angeboten die Arbeitspakete der TUM zu übernehmen.

Die Übernahme der Arbeitspakete der TUM gestaltete sich schwierig. Die daraus resultierende Verzögerung führte dazu, dass wir eine kostenneutrale Projektverlängerung um 6 Monate beantragt haben, welche uns auch genehmigt wurde.

3 Wesentliche Ergebnisse sowie die Zusammenarbeit mit anderen Forschungseinrichtungen

Unser Projekt teilt sich grob in die Erforschung der Machbarkeit einer Virtualisierungslösung auf einem Mikrokern, dem technischen Teil, und der Untersuchung der Anwendbarkeit formaler Methoden zur Stärkung der Vertrauenswürdigkeit der Lösung, auf.

Auf der technischen Ebene ist das Forschungsvorhaben in fünf wesentliche Bereiche unterteilt.

1. Verbesserungen des Basissystems für den Hypervisorbetrieb
2. UVMM Servicewelt; Die Verwendbarkeit von Gast-Linux Infrastruktur für das Mikrokernsystem, sowie andere Gast-Systeme
3. Unterstützung schwer virtualisierbarer Betriebssysteme
4. Unterstützung von vorhandenen Cloud-Infrastruktursystemen
5. Snapshot und Restore

Dem schließen sich die Bereiche der formalen Methoden an:

1. Abstraktes Modell und Sicherheitseigenschaften
2. Erweiterung der Testinfrastruktur
3. Randomisierte Tests

Im Folgenden werden wir die Ergebnisse und Erkenntnisse der einzelnen Bereiche besprechen.

3.1 Verbesserungen des Basissystems für den Hypervisorbetrieb

Die in diesem Abschnitt besprochenenen Arbeiten sind im Projektantrag unter den Punkten „UVMM: Stabilisierung und Integration“, „UVMM: Unterstützung direkter Gerätezuweisung“ und UVMM: Integration von OpenBSD„ zu finden.

Das gewählte Mikrokernbetriebssystem hat sich in seiner Grundstruktur als geeignete Basis für einen Cloud-Hypervisor erwiesen. Darum waren an der zugrundeliegenden Systemarchitektur tatsächlich keine Änderungen notwendig.

Einzelne Komponenten, insbesondere in der Virtualisierungsinfrastruktur, lagen zu Projektbeginn in einer sehr einfachen Version vor, der wichtige Fähigkeiten fehlten. Diese konnten im Projektverlauf implementiert werden. Beispiele dafür sind Verbesserungen in der Emulation von Plattformgeräten wie RTC, PIT, PIC, APIC und IOAPIC. Des Weiteren konnten wir die Fähigkeit zum exklusiven Durchreichen von Peripheriegeräten wie Netzwerkkarten umsetzen. Dabei wird ein Hardwaregerät derartig für eine virtuelle Maschine bereitgestellt, dass die Treiber des Gastbetriebssystems die Hardware direkt treiben, und nicht die Treiber des Host-Betriebssystems.

Für die Arbeiten unseres Konsortialpartners Genua haben wir diverse Anpassungen zur Unterstützung des Gastbetriebssystems OpenBSD vorgenommen. Dazu zählen Unterstützung bei der Einarbeitungen in die Details unserer Virtualisierungslösung, Hilfestellungen bei aufkommenden Fragen, und diverse technische Änderungen, sofern sie notwendig wurden. Beispielsweise wurde für OpenBSD eine Unterstützung für das Modifizieren der PCI base address registers (BARs) implementiert. Auch gab es diverse Anpassungen in der Interrupt-Behandlung durch den Hypervisor.

Durch die im Folgenden beschriebenen Arbeitspakete kamen stets neue Anforderungen auf, so dass wir stetig an der Verbesserung des Basissystems arbeiten mussten. Daher lässt sich im Nachhinein konstatieren dass der im Projektantrag veranschlagte Aufwand optimistisch angesetzt war.

3.2 UVMM Servicewelt

Dieses Themengebiet umfasst die Arbeitspakete AP MV4 “Identifikation von Gastdiensten und Protokolle„, AP MV5 “Beispielhafte Implementierung eines Dienstes„ und AP MV6 “Integration eines Services in den Microhypervisor„.

Ziel ist es, Dienste eines Gastsystems für andere Gastsysteme beziehungsweise das Mikrokernsystem bereitzustellen. Im Projektantrag haben wir uns auf persistenten Speicher, also Festplatten, beschränkt, da dies eine zwingende Voraussetzung für den Betrieb jeglicher virtueller Maschinen darstellt. Ohne die Möglichkeit von persistentem Speicher zu booten und berechnete Daten abzulegen, hätten virtuelle Maschinen wenig bis keinen Nutzen.

Im Mikrokernsystem wird als wichtigster Emulationsmechanismus virtio eingesetzt. So existieren bereits virtuelle Gerätemodelle für Netzwerk und Konsole. Virtio ist eine für Virtualisierung optimierte Geräteschnittstelle mit geringer Komplexität und exzellenter Leistungsfähigkeit. Es wäre daher wünschenswert, wenn der umzusetzende Dienst evenfalls über virtio abgebildet würde. Die Unterstützung von virtio im Mikrokernsystem ist als ausgereift und stabil anzusehen.

Während der Evaluierung möglicher Lösungen sind wir auf den vhost-user Mechanismus gestoßen. Dieser entstammt der Entwicklercommunity der Virtualisierungslösung KVM, und wurde dort in erster Linie für eine effiziente Netzwerkemulation entwickelt. Technisch wird dabei auf virtio als Transportmechanismus gesetzt, wobei die eigentlichen Gerätemodelle als Linux-Nutzeranwendungen implementiert werden. Es sind einige bereits sehr ausgereifte Gerätemodelle als Open Source verfügbar. Hier ergab sich also die Möglichkeit mit der Unterstützung eines einzigen Mechanismus, des vhost-user, die Funktionalität mehrerer Dienste umzusetzen.

Die Implementierung hat sich als komplex herausgestellt, so dass dieses Arbeitspaket mehr Aufwand gekostet hat als im ursprünglichen Projektplan vorgesehen. Es wurden zwei Komponenten entwickelt, welche die Basisfunktionalität abbilden. Zum einen wurde ein virtuelles Gerät in unserer Virtualisierungskomponente Uvmm implementiert. Dieses implementiert die Schnittstelle von der virtuellen Maschine zum Mikrokernsystem. Zum anderen wurde ein Linux Kernel-Modul und eine dazu passende Anwendung implementiert, mit der die Verbindung ins Mikrokernsystem von seiten des Gastsystems gesteuert werden kann.

Wir konnten im abschließenden Demonstrator die vhost-user Funktionalität mit einem Festplattenbenchmark zeigen.

3.3 Unterstützung schwer virtualisierbarer Betriebssysteme

Dieses Themengebiet umfasst die Arbeitspakete AP VM7 “Exploration UEFI Unterstützung„, AP MV8 “Exploration Unterstützung für geschlossene Systeme„, AP MV9 Emulation von Gerätemodellen“, AP MV10 „Messen und Optimieren der Geräteemulation“.

Im Projektantrag haben wir insbesondere auf das proprietäre aber umso verbreitetere Betriebssystem Windows 10 abgezielt. Im Projektverlauf hat sich gezeigt, dass Windows besonders schwierig virtualisierbar ist. Windows unterstützt nur eine proprietäre Virtualisierungsschnittstelle, hyperv, nativ. Virtio-Unterstützung kann mit Drittherstellern nach der Installation nachgerüstet werden. Windows hat sich als äußerst verschlossen und schwierig analysierbar gezeigt. So sind weder der Bootprozess noch Systemvoraussetzungen öffentlich dokumentiert und können ohne Einblick in den Quelltext auch nur schwer in Erfahrung gebracht werden.

Im Rahmen der Recherche haben wir einen Hypervisor gefunden, der Windows 10 von Haus aus hervorragend unterstützt: der Linux-basierte Hypervisor KVM.

So haben wir die Idee der verschachtelten Virtualisierung verfolgt. Dabei kann ein Hypervisor selber in einer virtuellen Maschine ausgeführt werden. So können wir KVM dazu benutzen schwer virtualisierbare Betriebssysteme, wie Windows, auf dem Mikrokernsystem zu virtualisieren.

Im Rahmen der Forschungsarbeiten haben wir die dazu notwendigen Fähigkeiten identifiziert und in der Virtualisierungskomponente UVMM implementiert. Erste Messungen zeigten jedoch eine völlig unzureichende Leistungsfähigkeit der Lösung. Im Folgenden haben wir untersucht an welchen Stellen der Mehraufwand dieser Lösung anfällt und haben diverse Optimierungen untersucht. Dazu musste die Mikrokernschnittstelle für die Virtualisierung und auch unsere Virtualisierungskomponente UVMM umgebaut werden. Zum Projektende konnten wir dadurch eine verschachtelte virtuelle Maschine in unserem Demonstrator zeigen, die eine akzeptable Leistungsfähigkeit zeigt.

3.4 Unterstützung von vorhandenen Cloud-Infrastruktursystemen

Die Arbeiten dieses Themengebiets sind im Projektantrag in den Arbeitspaketen AP MV11 „Analyse und Auswahl von Hypervisor API und Containerformat“, AP MV12 „Implementierung Hypervisor API“, AP MV13 „Integration von Tooling zur Provisionierung von virtuellen Maschinen“ zu finden.

Die in den vorherigen Themengebieten erarbeitete Funktionalität des Hypervisors muss auch von einer entsprechenden Cloud-Managementsoftware ansprechbar und damit nutzbar sein. Die Auswahl und Unterstützung einer entsprechenden API ist die Aufgabe dieses Themengebiets.

Unsere Recherchen haben uns auf die libVirt-API gebracht. Diese API ist als Open Source frei verfügbar. Sie dient als generische Abstraktionsebene, auf der verschiedene Cloud-Managementlösungen aufsetzten können. Beispiele für solche Lösungen sind OpenStack, virt-manager und virsh. Auf der anderen Seite unterstützt libVirt diverse Hypervisoren, wie z.B. KVM, VMware ESX, HyperV, Bhyve und Xen. Die Unterstützung für die einzelnen Hypervisoren ist gekapselt in entsprechenden Hypervisor-Treibern.

Wir haben also einen entsprechenden Hypervisor-Treiber für unseren Mikrokern-basierten Hypervisor implementiert.

Die korrekte Funktionalität konnten wir auf unserem abschließenden Konsortialtreffen in unserem Demonstrator zeigen.

3.5 Snapshot und Restore

In diesem Themengebiet hat Kernkonzept die theoretischen Vorarbeiten der Technischen Universität München (TUM) bekommen. So konnten wir uns im Wesentlichen auf die Implementierung des Demonstrators konzentrieren. Die hier beschriebenen Inhalte entsprechen folgenden Arbeitspaketen des Projektantrags: AP SR70 „Einarbeitung in Snapshot und Restore“, AP SR46 „Erstellung

eines Konzepts für das Restoring“, AP SR48 „Implementierung der Snapshot Funktionalität“, AP SR49 „Implementierung der Restore-Funktionalität“, AP SR50 „Evaluation der S/R Komponenten“, AP MV6 „Integration eines Services in den Microhypervisor“ und AP MV15 „Demonstrator Virtualisierungsserver“.

Wir haben uns entsprechend den Ergebnissen der TUM auf einen Snapshot-Mechanismus beschränkt, der unter Mitwirkung des Gastsystems funktioniert. Der Mechanismus besteht aus mehreren Komponenten. Zum einen haben wir den bereits beschriebenen vhost-user Mechanismus für eine Komponente benutzt, die ein persistieren von Daten ermöglicht. Des Weiteren wurde unsere Virtualisierungskomponente UVMM um folgende Funktionalitäten erweitert: Es gibt jetzt einen Mechanismus, mit dem von einer Steuerkomponente aus ein Signal an das Gastsystem geschickt werden kann. Dieses Signal veranlasst das Gastsystem ein Suspend to RAM zu initieren und durchzuführen. Wir haben unsere Virtualisierungskomponente um die Funktionalität des ACPI S3 Suspend sowie das dazu passende Resume erweitert.

Suspend to RAM ist ein Betriebssystemmechanismus. Dabei friert das Betriebssystem Anwendungen ein, schaltet Peripheriegeräte aus und instruiert die Firmware den Computer in den Suspendmodus zu schalten. Dabei wird der Computer so weit ausgeschaltet, dass CPU und Peripheriegeräte stromlos geschaltet werden und damit ihren gesamten Kontext verlieren. Einzig der Arbeitsspeicher wird weiterhin mit Strom versorgt und behält seine Inhalte. In unserer Virtualisierungslösung implementiert die Virtualisierungskomponente UVMM den Suspendmodus.

Wenn Suspend to RAM abgeschlossen ist, persistiert UVMM den Gastzustand aus dem Gast-Arbeitsspeicher sowie den vom Gastsystem vorgegeben Resume Vektor, also die Speicheradresse, an der das Gastsystem wieder aufgeweckt werden möchte. Nach dem Persistieren des Gastzustands wird die Virtualisierungskomponente UVMM beendet, und die virtuelle Maschine sowie deren Resourcen werden frei gegeben.

Das Restore wird anschließend durch einen frischen Start der Virtualisierungskomponente UVMM versehen mit einem zusätzlichen Kommandozeilenparameter durchgeführt. UVMM legt dabei eine leere virtuelle Maschine an, kopiert den Gastarbeitsspeicherinhalt in den Gastspeicher, und startet das Gastsystem am Resume Vektor.

Wir konnten Snapshot und Restore auf unserem Abschlusstreffen in unserem Demonstrator zeigen.

3.6 Abstraktes Modell und Sicherheitseigenschaften

Dieses Themengebiet umfasste das Erstellen eines abstrakten Modells von L4Re, das Erarbeiten und Beweisen von Sicherheitseigenschaften, und die Entwicklung eines automatischen Zustandsvergleichs zwischen Modell und echtem System. Die fraglichen Arbeitspakete sind FM 26 „Abs-

traktes L4Re-Modell“, FM 27 „Verhaltens- und Zustandsvergleich“, FM 31 „Abstraktes Modell im Theorembeweiser“ und FM 32 „Formaler Beweis der Sicherheitseigenschaften“.

Als Modell verstehen wir eine vereinfachte Simulation von L4Re, die insoweit vollständig sein soll, als das Programme, die in der Simulation und auf dem echten System laufen, sich gleich verhalten. Jede Abweichung weist auf einen Fehler im Modell oder dem echten System, L4Re, hin. Der Einsatz eines Modells bringt mehrere Vorteile mit sich: Bei seiner Implementierung müssen unklares Verhalten und „dunkle Ecken“ in L4Re bedacht und durchleuchtet werden. Dies führte tatsächlich zur Entdeckung und Behebung einer Reihe von Problemen. Weiterhin kann das Modell in einen Theorembeweiser übertragen werden, da es in einer funktionalen Sprache (Haskell), welche leichter für formale Methoden zugänglich ist, geschrieben ist. Im Theorembeweiser können Sicherheitseigenschaften mathematisiert, präzise formuliert und bewiesen werden. Zuletzt kann man aus dem Modell Testprogramme generieren und ihr Verhalten mit dem echten System vergleichen.

Im Rahmen von Versecloud konnte das Modell soweit entwickelt werden, dass es die grundlegende Funktionalität von L4Re abbildet. Dabei wurde der Fokus auf das Objektsystem gelegt. Eine vollständige Modellierung konnte allerdings nicht erreicht werden. Dieses Modell wird, wie weiter unten beschrieben, zur automatischen Testgenerierung verwendet. Es wurde außerdem in einen Theorembeweiser übertragen und folgende Sicherheitseigenschaft bewiesen: Wenn zwei Programme keinen Capabilitykanal zueinander haben, dann können sie auch keinen erzeugen. Bildlich gesprochen heißt das, dass, wenn sie voneinander isoliert sind, sie das auch bleiben. Dies ist eine grundlegende Eigenschaft, die L4Re aufweisen muss, damit Programme gegen möglicherweise bösartige andere Programme geschützt sind.

3.7 Erweiterung der Testinfrastruktur

Um vom Modell generierte Tests auf L4Re laufen zu lassen und ihre Ausführung wie beschrieben zu vergleichen, musste Vorarbeit geleistet werden. Dies umfasst die Arbeitspakete FM 23 „Auslesen und Vergleichen der Capabilityverteilung“, FM 24 „Extraktion und Ausführung abstrakter Testbeschreibungen“ und FM 25 „Erweiterung der Tests durch Testannotationen“.

Dabei wurde Infrastruktur geschaffen, um den internen Zustand von L4Re auszulesen und auszuwerten. Bestehende, händisch geschriebene Tests wurden erweitert um mithilfe derer den erwarteten Zustand mit dem Tatsächlichen während der Testausführung zu vergleichen. Dieses Vorgehen ist als Vorstufe zum Vergleich mit randomisierten Tests zu sehen, brachte aber dennoch bereits einen Gewinn in Form von aufgefundenen Problemen im L4Re.

Als weitere Vorarbeit wurde eine abstrakte Testspezifikation erarbeitet, in deren Form die randomisierten Tests generiert werden sollen. Als Probe ihrer Zulänglichkeit wurden sowohl per Hand abstrakte Tests geschrieben, als auch in nichtrandomisierter Form generiert. Auch hierbei wurden eine Reihe von Problemen in L4Re gefunden und behoben.

3.8 Randomisierte Tests

Dieser Bereich umfasste die Arbeitspakete FM 28 „Framework für modellbasiertes Testen“, FM 29 „Randomisierte Generierung von L4Re-Tests“ und FM 30 „Mängel- und Fehlerbehebung in L4Re“. Er beinhaltet alle abschließenden Arbeiten, um aus dem erstellten Modell randomisierte Tests zu generieren, auszuführen und gefundene Probleme in L4Re zu beheben. (Letzteres geschah auch bereits bei den beschriebenen Vorarbeiten.)

Zuerst musste hierfür eine Ausführungsumgebung für die abstrakten Testspezifikationen in L4Re aufgesetzt werden. Die Spezifikationen werden dabei automatisiert in in C++-Programme übersetzt, die mit erstellten Bibliotheken als native L4Re-Programme ausgeführt werden. Ähnlich wie die händisch erweiterten Tests geben sie den internen Systemzustand aus, damit dieser mit der Vorhersage des Modells verglichen werden kann. Im Gegensatz zu den händischen Tests können nun aber durch die Randomisierung große Parameterbereiche der L4Re-API automatisch erforscht und so die Implementierung auf Fehler überprüft werden.

Dieses maschinelle Testen erfolgt auf dem eigens für das Projekt angeschafften 96-Kern-Rechner in der Größenordnung von Millionen von Tests und sorgt so für ein wesentlich erhöhtes Vertrauen in die Sicherheit und Zuverlässigkeit von L4Re.

Die Hauptrichtung von Versecloud ausgehend diese Ergebnisse zu verbessern, muss darin bestehen, das Modell sukzessive zu verfeinern, ihm also den gleichen Funktionsumfang wie das echte System zu geben, und mehr Sicherheitseigenschaften zu beweisen.

3.9 TODO die wichtigsten Positionen des zahlenmäßigen Nachweises

3.10 die Notwendigkeit und Angemessenheit der geleisteten Projektarbeiten

Die durchgeführten Arbeiten sowie die dafür aufgewendeten Ressourcen waren notwendig und angemessen. Die bewilligten Mittel wurden ausschließlich für Personal- und Reisekosten verwendet. Da ohne entsprechendes Personal das Projekt nicht hätte bearbeitet werden können, waren die Mittel notwendig. Es wurden alle Arbeitspakete bearbeitet, daher war die geleistete Arbeit angemessen.

3.11 Voraussichtlicher Nutzen im Sinne des fortgeschriebenen Verwertungsplans

Die im Rahmen des Projekts durchgeführten Arbeiten erweitern das von Kernkonzept entwickelte und gepflegte Open Source Betriebssystem um wichtige Funktionalität. Diese zeigt, dass L4Re grundsätzlich als Basis für einen hochsicheren Cloudhypervisor dienen kann. Insbesondere die Verbesserungen bei der Benutzung der Hardwarevirtualisierungsschnittstelle stellen für Kernkonzept ein wichtiges Asset dar, welche für zukünftige Entwicklungen mit L4Re den Weg frei machen.

3.12 Dem Zuwendungsempfänger bekannt gewordener Fortschritt auf dem Gebiet des Vorhabens bei anderen Stellen

Sowohl auf der technischen als auch auf der Seite der formalen Methoden sind uns keine für das Projekt relevanten Entwicklungen bekannt.

3.13 die erfolgten oder geplanten Veröffentlichungen des Ergebnisses nach Nr. 5 der NKBF/NABF

Im Projekt Versecloud sind die folgenden Veröffentlichungen erfolgt:

1. “Eine vertrauenswürdige, sichere Public Cloud – Utopie oder Realität?”

   (a) Deutscher IT-Sicherheitskongress des BSI 2023, digital

2. “VerSeCloud – Der Weg zur vertrauenswürdigen und sicheren Cloud ‘Made in Germany’ ”

   Behörden Spiegel / April 2023

3. “Bringing Synthesised Software to a Real-World Microkernel Operating System”

   The 35th Symposium on Implementation and Application of Functional Languages (2023)

   Braga, Portugal

4. “A Development Process for Coq Projects Permitting Invalid Proofs”

   The Coq Workshop 2024

   Tbilisi, Georgien

Es wurden von Kernkonzept keine gewerblichen Schutzrechte angemeldet.

Kurzbericht VerseCloud

Kernkonzept GmbH

11. Dezember 2024

- ZE: Kernkonzept GmbH
- Förderkennzeichen: 16KIS1357K
- Vorhabenbezeichnung: Microhypervisor für sicherheitskritische Cloudanwendungen mit formal verifiziertem Design
- Laufzeit des Vorhabens: 01.04.2021 - 30.09.2024

1 Aufgabenstellung und Stand der Wissenschaft/Technik

Wie können Mikrokerne zu einem performanten, flexiblen und vertrauenswürdigen Hypervisor mit sehr hoher Konfidenz in die Korrektheit seine Implementierung weiterentwickelt und um integritätsgeschützte Snapshotting-Werkzeuge erweitert werden?

2 Ablauf des Vorhabens

- Projektstart: 01.04.2021
- Projektende: 30.09.2024

Ursprünglich war das Projekt auf eine Laufzeit von 3 Jahren ausgelegt. Nach einem Projektjahr ist der Konsortialpartner Technische Universität München aus dem Projekt ausgeschieden. Durch diese Änderung kam es zu Verzögerungen und in der Folge zu einer Verlängerung der Projektlaufzeit um 6 Monate.

3 Wesentlichen Ergebnisse sowie ggf. die Zusammenarbeit mit anderen Forschungseinrichtungen

In diesem Projekt konnten wir zeigen dass es technisch möglich ist auf einem Mikrokernsystem einen Cloudfähigen Hypervisor aufzubauen, und es wurden erste Schritte gemacht dessen Sicherheit mit formalen Methoden nachzuweisen. Dazu wurde die Virtualisierungsinfrastruktur des Mikrokernsystems verbessert und für die Cloud wesentliche Bestandteile erweitert werden. Es wurde ein effizienter Mechanismus geschaffen um Dienste von einem Gastbetriebssystem für andere Gastsysteme sowie das Mikrokernsystem zu erbringen. Wir konnten zeigen dass schwer virtua-

lisierbare Betriebssysteme wie Windows 10 auf dem Mikrokernsystem lauffähig sind. Mit libvirt konnten wir eine Cloudfähige API für unsere Virtualisierungslösung finden und zeigen, dass sie mit marktüblicher Cloudmanagementsoftware integrierbar ist. Aufbauend auf den theoretischen Erkenntnissen unseres Konsortialpartners Technische Universität München konnten wir einen Mechanismus für Snapshot und Restore implementieren, der Snapshots mit besonders hohen Integritätsanforderungen ermöglicht. Die genannten Funktionalitäten haben wir abschließend in einen Demonstrator integriert und auf dem finalen Abschlusstreffen gezeigt.

Mittels formaler Methoden konnten wir ein Modell des Systems erstellen und für dieses wichtige Sicherheitseigenschaften beweisen. Mittels dieses Modells konnten wir eine maschinelle Testumgebung für den Hypervisor entwickeln, die zufallsgenerierte Tests nutzt, um eine hohe Vertrauenswürdigkeit in die Sicherheit und Zuverlässigkeit zu gewährleisten.

Unserem Partner Genua konnten wir Unterstützung für deren Arbeiten an einer Virtualisierung von OpenBSD leisten. Das Ergebnis dieser Arbeiten konnte Genua in einem Demonstrator zeigen.