ELEKTRONIK TIDNINGEN

EXPERTARTIKEL

Gernot Heiser
Teknikchef
Open Kernel Labs

Så byggde vi ett ofelbart OS

OK Labs har gjort ett formellt bevis för att företagets operativsystemskärna är korrekt. Det innebär ett forskningsgenombrott. Formella bevis går långt utöver de krav som idag ställs för exempelvis certifiering av säkerhetskritisk programvara. Professor Gernot Heiser berättar hur beviset togs fram och vad det innebär.

Redaktör
Jan Tångring
jan@etn.se
0734-17 13 09

EMBEDDED EXPERT

Så byggde vi ett ofelbart OS

OK Labs hypervisor saknar buggar = SANT

Av Gernot Heiser, Open Kernel Labs

Gernot Heiser är teknikchef på Open Kernel Labs som kommersialiserar mikrokärnan L4 – ett forskningsgenombrott från slutet av 90-talet som idag används för virtualisering i 300 miljoner mobiltelefoner, bland dem Sony Ericsson Xperia.

Förra året rapporterade Open Kernel Labs, universitetet i New South Wales och forskningcentret NICTA att ett gemensamt projekt till slut var fullbordat: att ta fram ett matematiskt bevis för korrektheten hos en generell mikrokärna och virtualiseringsplattform.

Beviset var kulmen på ett gemensamt forsknings- och utvecklingsprojekt med målet att ta fram utvecklingsmetoder för programvara som kan garantera extremt ställda krav på pålitlighet i tillämpningar med säkerhets- och sekretesskrav inom exempelvis flyg, nationell säkerhet och transport.

Våra bevismetoder kommer att göra det betydligt enklare att certifiera sig för Common Criteria och andra standarder för säkerhet och sekretess. Det banar också väg för användandet av virtualisering inom affärskritiska områden som mobiltelefoni, mobila finansiella transaktioner och business intelligence.

Dagens olika program för att certifiera programvara för kritiska tillämpningar, lägger fokus på systemutvecklingsprocesser, test och följandet av mjukvarumodeller. Vårt angreppssätt är att istället arbeta mot den resulterande mjukvaran och bevisa att den är korrekt med hjälp av formell logik och datorstödd satsbevisning.

Vi har blivit de första att kunna slutföra ett sådant bevis för en generell operativsystemskärna eller en hypervisor.

Verifieringsprocessen eliminerade ett antal buggar i kärnan som skulle kunna ha exploaterats – både konstruktionsmissar och kodbaserade fel som exempelvis buffertöverskrivning, nollpekarreferenser, minnesläckor och aritmetiskt spill.

Det tog oss fem år att verifiera 7500 kodrader och beviset krävde att 10 000 hjälpteorem bevisades i sammanlagt drygt 200 000 härledningssteg.

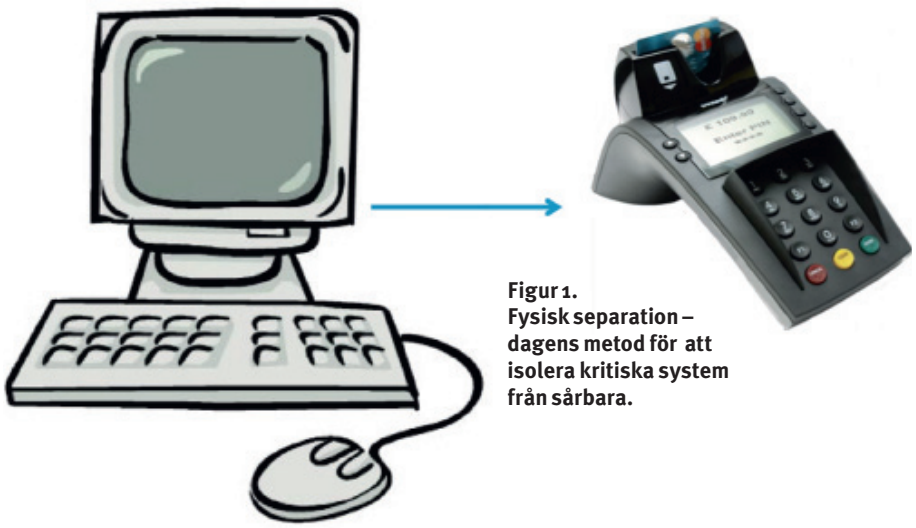
Den verifierade kodbasen en är mikrokärna av typen L4. Den utgör grunden för plattformen OK:Verified och kan användas som en hypervisor som kan vara värd för multipla virtuella maskiner.

Mikrokärnan är av den så kallade tredje generationen, och är influerad av projektet Eros (Extremely Reliable Operating System). Den stöder abstraktioner för virtuella adressrum, trådar, processkommunikation och acesskontroll av det slag som Eros har, men som saknas i de flesta kommersiella operativsystem.

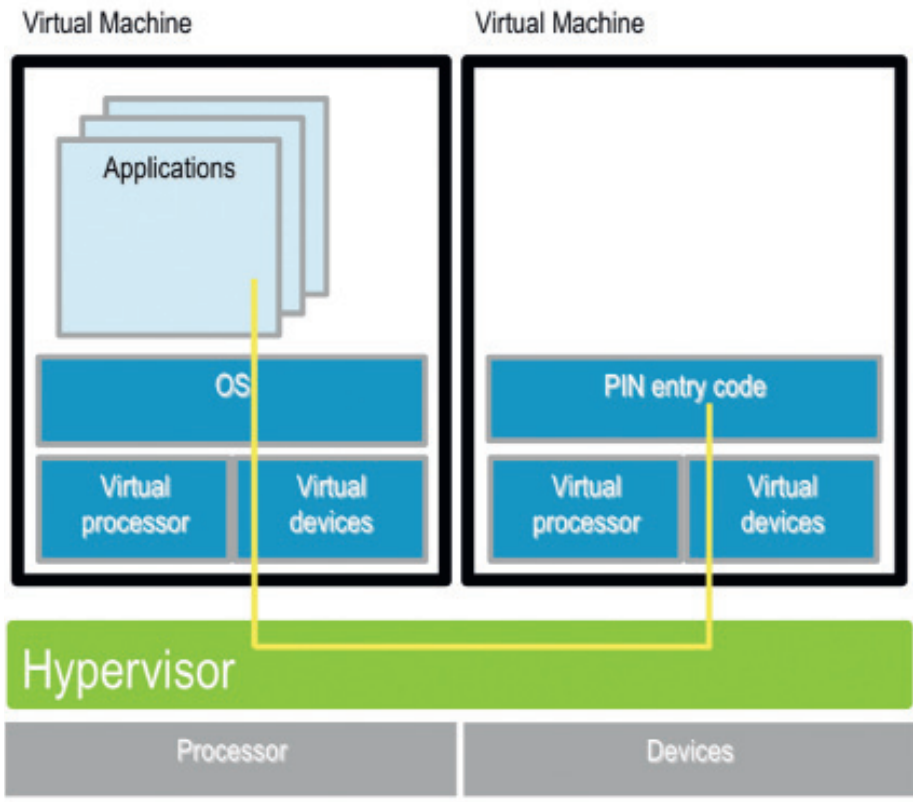
Utveckling och verifiering gjordes på ARMv6 varefter kärnan porterades till x86.

Vårt maskinrättade bevis är ett av de största som någonsin slutförts. Det finns andra formellt bevisade kärnor, men vårt projekt saknar motsvarighet för mjukvaror av motsvarande storlek och komplexitet.

För att kunna nå den här milstolpen var vi tvungna att konstruera nya metoder för maskinbevisning, för programspråksbeskrivning, och för prototypframställning av operativsystem. Kärnan är dels bevisligen fri från ett antal specifika fel, och dels har vi verifierat att kärnan i sin helhet överensstämmer med sin systemspecifikation. Utöver själva mikrokärnan har vi fått som resultat en uppsättning generella metoder för att utveckla och verifiera mjukvara.



Figur 1.
Fysisk separation – dagens metod för att isolera kritiska system från sårbara.



Figur 2.
En hypervisor som vill isolera lika bra som fysisk separation, kan inte ha säkerhetshål.

Kodstorlek och komplexitet är svurna fiender till pålitlighet. Även i mycket noggrannt konstruerad mjukvara finns mellan två och fem buggar per tusen kodrader (kLoc) vilket är ett stort bekymmer ur säkerhets- och sekretessynpunkt. Mobila trådlösa apparater består miljontals kodrader och har därmed tusentals inneboende sårbarheter.

Säkerhet och pålitlighet i ett datorsystem grundas på den underliggande kärnan, definierad som de delar av systemet som exekveras med maximala privilegier – och exempelvis obegränsad access till hårdvaran. Ett implementeringsfel här underminerar hela systemet.

Det är omöjligt att helt undvika buggar i större kodbaser. När säkerhet och pålitlighet är av största vikt är det därför starkt rekommenderat att den privilegierade koden görs så begränsad som möjligt – då minimeras sårbarheten för buggar. Det här är ett sätt att möta utmaningen att kombinera komplexitet med robusthet.

De delar av systemet som kan åsidosätta säkerhetspolicies kallas med ett namn för Trusted Computing Base (TCB). En minimal TCB leder till större pålitlighet.

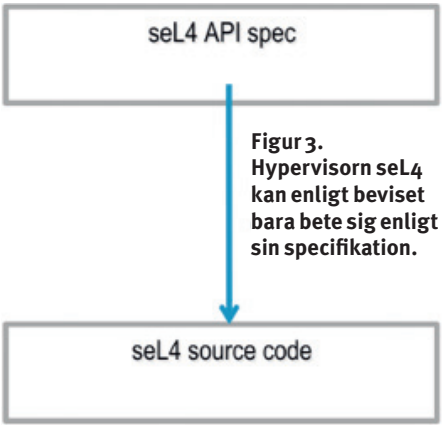
Fast det är lättare sagt än gjort – rika användargränssnitt och tillämpningsprogram körs ofta i operativsystem som Windows, Linux eller Android. De är stora och komplexa och därmed fulla av buggar. Problemet är att operativsystemet ofrånkomligen är en del TCB:n.

Virtualisering är en elegant lösning för TCB-reduktion. Det innebär att man delar upp systemet i separerade virtuella maskiner (VM). I exempelvis en mobil kan en VM köra ett rikt OS med användarprogram medan en annan kör ett realtidsoperativsystem (RTOS) för trådlös kommunikation och en tredje kör säkerhetskritisk kod. TCB:n kan därmed till övervägande del bestå av en så kallad hypervisor som administrerar säkerhet och sekretess.

En hypervisor för mobila krav kan byggas i mindre än 10 kLoc. Antalet buggar i en så liten kodbas kan göras hyfsat litet med hjälp av etablerad god utvecklingspraktik.

Efteråt kan man hålla tummarna för att ingen av de kvarvarande buggarna gör systemet sårbart.

Men när människoliv och rikets säkerhet står på spel, borde vi egentligen ha en mer pessimistisk approach och göra antagandet att där faktiskt finns kritiska buggar. Tills motsatsen bevisats.



Figur 3.
Hypervisorn seL4 kan enligt beviset bara bete sig enligt sin specifikation.

För den som verkligen vill garantera att ett system är buggfritt är formell verifiering det enda existerande alternativet till en komplett genomtestning. Formell verifiering är dock en ganska okänd teknik och används sällan – den anses dyr och användbar endast för mindre kodbaser. Men som vi nu demonstrerat är den faktiskt ett realistiskt alternativ för en mindre mikrokärna.

Vår metod var att göra stegvis förfining med hjälp av en interaktiv teorembevisare. Förfiningen leder i bevis att en abstrakt högnivåmodell och en konkret lågnivåmodell motsvarar varandra – alla tänkbara beteenden hos den konkreta modellen fångas av den abstrakta modellen.

Den abstrakta modellen specificerar kärnans beteende – dess programgränssnitt – uttryckt i ett matematiskt språk kallat högre ordningens logik (HOL). Den konkreta modellen är implementeringen – kod som översätts av en kompilator till en exekverbar fil. Förfiningen visar att kärnan beter sig i enlighet med sin specifikation, med andra ord att kärnan är funktionellt korrekt.

Beviset för funktionell korrekthet omfattar klassiska säkerhetsegenskaper som att kärnan inte kan krascha eller utföra osäkra operationer.

Funktionell korrekthet innebär att det är möjligt att förutse kärnans reaktion på varje tänkbar situation – därmed kommer aldrig någonting oväntat att inträffa. Det betyder exempelvis att det är omöjligt att injicera godtycklig kod i kärnan, som sker vid så kallade ”stack-smash”-attacker.

Det faktiska beviset utfördes och kontrollerades av en interaktiv teorembevisare kallad Isabelle/HOL. Översättningen från C till HOL är kritisk för bevisets korrekthet; vår forskargrupp modellerade exakt enligt standarden vad gäller C:s semantik, typapparat och minnesmodell, med exempelvis processorberoende ordlängd, utfyllnadsbitar i structar, typosäker pekarkonvertering och pekararitmetik.

HOL kräver mänsklig medverkan och kreativitet för att konstruera och vägleda beviset. HOL är å andra sidan inte begränsad till att hantera vissa specifika egenskaper i ändliga tillståndsrum, till skillnad från andra mer helautomatiska verifieringsmetoder som statisk analys och modellkontroll.

För underlätta certifiering måste komplexiteten i kärnans komponenter minimeras. Idealt skulle man önska sig kod (och bevis) för kärnan i form av triviala satser avhängiga endast explicita

lokala tillstånd och triviala invarianter. Olyckligtvis finns i kärnor ofta beroenden mellan delsystem. Genom att ta bort det ur kärnan som istället kan implementeras som användarprogram stängs operativsystemets ”rörighet” in, med omfattande länkar mellan dess komponenter.

OS-utvecklare har en nerifrån-och-upp-ansats till kärnkonstruktion – man vill krama ut prestanda ur hårdvaran. Detta leder till konstruktioner som är fokuserade på lågnivådetaljer.

Formalister jobbar tvärtom uppifrån och ner eftersom bevisens spårbarhet beror av systemets komplexitet. Det resulterar i enklare modeller, abstraherade från hårdvaran.

Vi blandade båda synsätten – till gläjde för både OS-utvecklare och formalister – och använde det funktionella programspråket Haskell för utveckling, automatöversättning och teorembevisning.

För att kunna exekvera Haskellprototypen realistiskt, länkade vi den till QEMU, en snabb emulator där användarprogram kan köras samtidigt som felavbrott slussas vidare till modellen av kärnan. I prototypen simuleras privilegierad exekvering i kärnan genom att användarprogramtillstånd modifieras.

Ofta reses frågan om korrektheten hos beviset i sig. Det finns flera exempel på att inkorrekta bevis publicerats i den matematiska litteraturen. Kan det inte ha uppstått fel också i våra bevis?

Traditionella bevis med penna och papper är känsliga för den mänskliga faktorn – skrivfel, förbiseenden och felaktiga eller ofullständiga bevisomskrivningar. Teorembevisare undviker sådana fallgropar – de kan endast applicera de transformationer som den formella logiken tillåter.

Bevisets korrekthet beror i princip också på den matematiska logikens lagar och på bevismaskinens kärna, bevisverifieraren. Båda är helt oberoende av denna tillämpning och har använts i en rad tillämpningar tidigare. Ansatsen som sådan är resultatet av 30 års teorembevisning. En ansats av detta slag kan idag uppnå den pålitlighet som krävs för formella maskinkontrollerade bevis där konfidensnivåerna anpassas till rimliga nivåer för säkerhetskritiska och livskritiska system. Från teorembevisaren Isabelle kan du dessutom exportera bevisrepresentation för användning i andra verifierare.

Ytterligare antaganden som vi gör är att de kompilatorer vi använder är korrekta, att några korta sekvenser assembler på kärnnivån är korrekta och att hårdvaran är korrekt. Var och en dessa antaganden kan om det är felaktigt leda till att kärnan fungerar inkorrekt trots att implementeringen är korrekt.

En verifierad kompilator skulle kunna eliminera kompilatorberoendet. En sådan kompilator för ARM har utvecklats av franska INRIA.

Att vi litar på assemblerkoden i kärnan är en tillfällig restriktion – verifiering av assemblersekvenserna kommer snart att presenteras.

De flesta kostnaderna för projektet var engångskostnader. Idag, efteråt, skulle vi kunna konstruera, implementera och verifiera liknande system för en kostnad en magnitud under typiska utvecklings- och QA-kostnader för Common Criteriaprogramvara, och dessutom erbjuda starkare garantier. Det inkluderar konstruktion och implementation av kärnan, utveckling av bevisverktyg, teknik, bibliotek och själva beviset.

Det existerar en komplett forskningsrapport (*Formal Verification of an OS Kernel*) för den som vill veta mer om kärnans konstruktion, om exempelvis införandet av globala variabler, minneshantering, jämlöpande processer, och in- och utmatning. ■

Denna artikel har också publicerats i magasinet Elektroniktidningen, nummer 3, 2010.