Minimize your TCB using a Microkernel-Based System

Udo Steinberg
Agenda

❖ The Fundamental Flaw in Today’s Security Model
❖ Building a Trustworthy Trusted Computing Base
  ➢ Microkernel / Microhypervisor
  ➢ Capability-Based Access Control
  ➢ Formal Verification
  ➢ Active Security
❖ Advanced x86 Security Technologies
❖ Q & A
Trusted Computing Base

- “A small amount of software and hardware that security depends on and that we can distinguish from a much larger amount that can misbehave without affecting security” (B. Lampson)

- From a security perspective it is desirable to
  - Minimize the Trusted Computing Base (TCB)
  - Implement Fine-Grain Functional Disaggregation (Modularity)
  - Enforce the Principle of Least Authority (POLA)

- Size of the TCB is application-specific
The Fundamental Flaw in Today’s Security Model

- Significant parts of the code base are trusted, but not trustworthy
  - Millions of SLOC in modern kernels, ⅔ of it in device drivers (Linux 6.8: ~25 million)
- Huge attack surface for code running with highest execution privileges
  - Security controls can be silently disarmed because they run at the same privilege level that they are trying to protect
Virtualization / Operating System Encapsulation

❖ Using virtualization replaces physical with logical isolation
❖ Hypervisor layer increases the TCB size further
❖ Existing security problems move one layer down
❖ What have we gained?
Summary: Castles Built on a Foundation of Sand

- Complex systems software with exploitable security vulnerabilities
- Defenders operate at the same privilege level as attackers
- Contemporary security software can be subverted by kernel-mode malware
- Traditional security model is failing against advanced attacks
BedRock Systems

Next-Generation Workload & Runtime Security
BedRock Systems

❖ Silicon Valley Based, Venture Capital Funded Startup
  ➢ Highly distributed: HQ in San Francisco, offices in Boston, Germany, Bangalore, …

❖ Operating Systems Experts
  ➢ Building a very small and trustworthy TCB (around the NOVA Microhypervisor)

❖ Formal Methods Experts
  ➢ Proving mathematically that the BedRock TCB conforms to its specifications

❖ Security Experts
  ➢ Using the BedRock TCB to introspect and harden VMs and container runtimes
Making the TCB Trustworthy

- Using a Microkernel instead of a Monolithic Kernel
  - Reduces the TCB size by more than 2 orders of magnitude
  - Enforces modularity and well-defined interfaces ⇒ Formal Verification becomes feasible

Using a Microkernel instead of a Monolithic Kernel:

- Reduces the TCB size by more than 2 orders of magnitude
- Enforces modularity and well-defined interfaces ⇒ Formal Verification becomes feasible
Microkernel Construction Principle

❖ “A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e. permitting competing implementations, would prevent the implementation of the system’s required functionality” (J. Liedtke)

❖ Design Goals
  ➢ Make the microkernel as small and fast as possible
  ➢ Provide only mechanisms (but no policies) in the microkernel
  ➢ Implement most system functionality in deprivileged user-mode components
  ➢ Enforce the principle of least authority among all user-mode components (zero trust)
NOVA: Portable Unified Code Base (x86/Arm)

<table>
<thead>
<tr>
<th></th>
<th>x86</th>
<th>x86_64-specific</th>
<th>generic</th>
<th>aarch64-specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLOC</td>
<td>13996</td>
<td>8789</td>
<td>5207</td>
<td>5801</td>
</tr>
<tr>
<td>ASM</td>
<td>4.2%</td>
<td></td>
<td></td>
<td>3.8%</td>
</tr>
<tr>
<td>Generic</td>
<td>37.2%</td>
<td></td>
<td></td>
<td>47.3%</td>
</tr>
</tbody>
</table>

NOVA x86 ELF Binary
- 86377 Bytes Code
- 2520 Bytes Data

NOVA Arm ELF Binary
- 77896 Bytes Code
- 328 Bytes Data

SLOC based on release-24.17.0, binary sizes based on gcc-13.2.0 build. Other versions will produce different numbers.
Microkernel Building Blocks

Before IPC Call

PD_A

EC_{Client} UTCB

PD_B

EC_{Server} UTCB

Before IPC Reply

PD_A

EC_{Client} UTCB

PD_B

EC_{Server} UTCB

Very fast synchronous IPC with time donation and priority inheritance

❖ Protection Domains, Execution/Scheduling Contexts, Portals, Semaphores

NOVA Microhypervisor
From Microkernel to Microhypervisor

- Microkernel interface is not POSIX-compliant
- Reuse of legacy operating systems via VMs
- Deprivileged Virtual-Machine Monitor (VMM)
  - VM exits are forwarded to the user-mode VMM for handling – instruction and device emulation
  - Per-event portal defines subset of architectural state that NOVA transmits to the VMM’s UTCB
  - VMM responds with updated state in its UTCB and optionally an event to inject
NOVA Microhypervisor Functionality

- Enumerates platform resources using UEFI/ACPI
- Manages security-critical functions of the platform
  - CPU, FPU, VMCS, MMU, SMMU (IOMMU), Interrupt Controllers (LAPIC, IOAPIC, GIC)
- Enforces spatial and temporal isolation between host components and VMs
  - Each component runs in its own address space
  - Preemptive fixed-priority round-robin core-local scheduler
- Provides very fast core-local communication via IPC

⇒ NOVA implements only mechanisms, but no policies
Capability-Based Access Control

- All syscalls based on capabilities
  - No designation without authority
  - No ambient authority
- Principle of least authority (POLA)
  - Components only possess capabilities for the resources they need
- Capabilities can be delegated
  - Permissions can be further restricted
BedRock Ultravisor Architecture

Formal Verification of Bare Metal Property™

VM
Linux

VM
Container Runtime

VM
Appliance

VM
RTOS

VM
Unikernel

VMM

UART Multiplexer
(Virtual Console Switch)

Network Multiplexer
(Virtual Ethernet Switch)

Host Applications

NOVA Microhypervisor
(ARMv8-A or Intel x86-64)

Master Controller
(Root Protection Domain)

UART Driver

Storage Driver

Platform Manager

Service Manager

Network Driver

Minimize your TCB using a Microkernel-Based System
Udo Steinberg - 2nd Charter of Trust Meetup Braunschweig
File-Modular Verification of Concurrent C++ Code using Separation Logic

Specifications can differ for disciplined vs. undisciplined components

```c++
/*
 * \arg{v1} “x” (Vint v1)
 * \arg{v2} “y” (Vint v2)
 * \post{}[Vint (trim 32 (v1 + v2))] emp
 */
auto add_func (uint32_t x, uint32_t y)
{
    return x + y;
}
```
Active Security: Fortify VMs & Container Runtimes

VM (Enlightened) -> VMM -> Active Security -> VMM -> VM (Non-Enlightened) -> VMM

VM (Non-Enlightened) -> VMM -> Active Security -> VMM -> VM (Unsecured) -> VMM

NOVA Microhypervisor (ARMv8-A or Intel x86-64)

Secure Vantage Point

Observe: Non-Bypassable Monitoring
Detect: Invisible Instrumentation
Protect: Software Hardening

Attackers with guest kernel privileges cannot evade or disarm the active security mechanisms implemented in the imperceptible Ultravisor layer.
Scaling NOVA from Embedded to Cloud Servers

---

Minimize your TCB using a Microkernel-Based System

Udo Steinberg - 2nd Charter of Trust Meetup Braunschweig

---

All trademarks and brand names are the property of their respective owners. All company and product names used in these slides are for identification purposes only. Use of these names, trademarks and brands does not imply endorsement.
Advanced x86 Security Technologies

Hardening the Platform Further
Multi-Key Total Memory Encryption (TME-MK)

- KeyID per page encoded in PTE
- Stealing upper physical bits

<table>
<thead>
<tr>
<th>Unused</th>
<th>KeyID</th>
<th>Physical Address</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>KeyID</td>
<td>Physical Address</td>
<td>Attributes</td>
</tr>
<tr>
<td>Unused</td>
<td>KeyID</td>
<td>Physical Address</td>
<td>Attributes</td>
</tr>
</tbody>
</table>

- Key Programming
  - random/tenant
  - DRNG entropy

Minimize your TCB using a Microkernel-Based System
Udo Steinberg - 2nd Charter of Trust Meetup Braunschweig
Protecting against “Noisy Neighbor” Domains

Minimize your TCB using a Microkernel-Based System

Udo Steinberg - 2nd Charter of Trust Meetup Braunschweig
Cache Allocation Technology (CAT/CDP)

Competitive Capacity Sharing  Exclusive Use

- COS 0: 35%
- COS 1: 25%
- COS 2: 20%
- COS 3: 30%
- COS 4: 50%
- COS 5: 15%

Minimize your TCB using a Microkernel-Based System
Udo Steinberg - 2nd Charter of Trust Meetup Braunschweig
Code Integrity Protection

- Long history of paging features raising the bar for code injection attacks
  - Non-writable code / Non-executable stack (W^X)
  - Supervisor Mode Execution Prevention (SMEP)
  - Supervisor Mode Access Prevention (SMAP)
  - Mode-Based Execution Control (MBEC) for Stage-2 with XU/XS permission bits

- Code snippets (gadgets) in existing code could still be chained together
  - Control-Flow Hijacking: COP / JOP / ROP attacks
  - Instruction length is fixed on ARM but varies on x86
Control-Flow Enforcement Technology (CET)

❖ Protects integrity of control-flow graph using x86 hardware features

❖ Indirect Branch Tracking (Forward-Edge) \textit{make ARCH=x86\_64 CFP=branch}
  ➢ Used with indirect JMP / CALL instructions
  ➢ Valid branch targets must be marked with ENDBR instruction
  ➢ Requires compiler support (available since gcc-8)

❖ Shadow Stacks (Backward-Edge) \textit{make ARCH=x86\_64 CFP=return}
  ➢ Used with CALL / RET instructions
  ➢ Second stack used exclusively for return addresses
  ➢ Can only be written by control-transfer and shadow-stack-management instructions
CET Indirect Branch Tracking

- CALL / JMP Instruction
  - Next instruction must be ENDBR
  - #CP exception otherwise

```
call *0x30(%rbx)  call *0x30(%rbx)
```

```
fffffffff80003a60 <Buddy::free(void*)>
endbr64

endbr64

test  %rdi,%rdi
je  ffffffff80003a84
sub  0x1e8(%rip),%rdi
shr  $0xc,%rdi
imul  $0x18,%rdi,%rdi
add  0xf1d1(%rip),%rdi
jmp  ffffffff80003962
ret
```
**CALL instruction**
- Pushes return address onto both stacks

**RET instruction**
- Pops return address from both stacks
- #CP exception if addresses not equal

**Shadow Stack Management**
- Busy bit in token prevents multi-activation
- NOVA must unwind supervisor shadow stack during context switches
Trusted Computing

- Once you have a formally verified software stack
  - and a compiler that produced a qualified set of binaries for the target architecture

- How do you ensure that some computer is running *those* binaries
  - and not some other (malicious) software instead
  - before you entrust that computer with your data or secrets

- In other words, how can you
  - either restrict the software that a computer will launch
  - or determine what software has been launched on a computer
Verified Boot: Static Root of Trust

❖ Boot policies are enforced during the boot process
❖ Starting with the Core Root of Trust for Verification, the currently executing module verifies the integrity of the next module against a boot policy (e.g. UEFI db/dbx) ⇒ Chain of Trust
❖ Integrity measurement is a cryptographic hash ⇒ unique + indicative to changes in the module
Measured Boot: Static Root of Trust

- Integrity measurements are extended into TPM PCRs during the boot process
- Starting with the Core Root of Trust for Measurement, the currently executing module extends the launch integrity measurement for the next module into the TPM
DRTM Flow lets system boot into an untrustworthy state (initially)

- Measured Launch later “resets” system into a trustworthy safe state
- Takes control of all CPUs and forces them down a protected and measured code path
Trusted Execution Technology: Measured Launch

- Design Decision: NOVA late-launches itself (~650 LOC)

- Unrecoverable failure causes TXT Shutdown

- Time

  - ILP
    - Load ACM
    - Load MLE
    - Launch (DRTM)
    - SENTER Event
  - RLPs
    - GETSEC[SENTER]
    - ILP broadcasts SENTER message
    - Each CPU responds to SENTER event
    - Each CPU issues ACK
    - ILP continues once all ACKs received
  - SENTER Event
  - SINIT ACM
  - Measured Launch Environment (NOVA)
  - MLE (NOVA)

- All CPUs in secure environment
- GETSEC[WAKEUP] for RLPs
- SINIT launches MLE
- ILP launches SINIT ACM
A verifier can use the crypto agile event log to recompute/validate the composite value in each PCR.
Confidential & Trusted Computing Building Blocks

❖ Availability
➢ Cache & Memory Bandwidth Allocation Technology (CAT/CDP/MBA)

❖ Integrity
➢ Control-Flow Enforcement Technology (CET IBT+SSS)

❖ Confidentiality
➢ Total Memory Encryption with Multiple Keys (TME-MK)

❖ Measured Launch & Attestation
➢ Trusted Execution Technology (TXT/CBnT)

Minimize your TCB using a Microkernel-Based System
Udo Steinberg - 2nd Charter of Trust Meetup Braunschweig
Questions and Discussion

The NOVA microhypervisor is licensed under GPLv2

Releases: https://github.com/udosteinberg/NOVA/tags

More Information: bedrocksystems.com and hypervisor.org