The Nested Kernel Architecture

1) What's the problem?

2) Nested Kernel Approach

3) Intra-Kernel Isolation

4) Evaluation

5) Future Work
Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation

*Nathan Dautenhahn  
Theodoros Kasampalis  
Will Dietz  
John Criswell  
Vikram Adve  

UIUC  
UIUC  
UIUC  
University of Rochester  
UIUC
Nested Kernel

Key Takeaways

- Retrofit the nested kernel architecture in FreeBSD 9.0
- Isolate MMU at a single hardware privilege level by virtualizing supervisor mode
- Intra-kernel memory protection services
- Lifetime kernel code integrity in FreeBSD 9.0
Monolithic Operating System

Outer Kernel

Nested Kernel

Write Protection Service

Virtualize Supervisor Mode
Key Takeaways

- Retrofit the nested kernel architecture in FreeBSD 9.0

- Isolate MMU at a single hardware privilege level by virtualizing supervisor mode

- Intra-kernel memory protection services

- Lifetime kernel code integrity in FreeBSD 9.0
The Nested Kernel Architecture

1) What's the problem?
2) Nested Kernel Approach
3) Intra-Kernel Isolation
4) Evaluation
5) Future Work
Monolithic Operating System Architecture

Drivers

File System

Virtual Memory

CPU Virtualization

Networking
Drivers

Monolithic Operating System Architecture

File System

Virtual Memory

CPU Virtualization

Networking
Simple malware are extremely powerful:

System Call Hooking

Problem: A complete lack of memory isolation
Code Injection

operation;
operation;
operation;

a NOOP d();
return;

Violates the integrity of audit recording
Simple malware are extremely powerful:

System Call Hooking

Problem: A complete lack of memory isolation
Observation: If we can Restrict Access to the Page Tables then Enable Memory Isolation

Page Tables in Typical Systems

CPU

Memory

Root Level

Enforce Access Control Policy

Translation

Modifiable
Super-Duper-Ooper-Schmooper Big Idea:
Isolate the MMU using the MMU

- Single Privilege Level
- Efficient Privilege Switch
- Apply to Monolithic OS
- No Control Flow Integrity
Virtualizing the MMU with the Nested Kernel

Threat Model
- Goal: restrict MMU access to nested kernel
- Outer Kernel Under Complete Control of Attackers
- Source Code
- Execution State
- No Control Flow Integrity
- Nested Kernel source is trusted

Nested Kernel Property
The nested kernel intercepts on all modifications of the MMU

The nested kernel isolates the MMU and provides lifetime kernel code integrity
Threat Model

Goal: restrict MMU access to nested kernel

- Outer Kernel Under Complete Control of Attacker
  - Source Code
  - Execution State
- No Control Flow Integrity
- Nested Kernel source is trusted

Isolate the MMU not defend against general kernel attacks
Nested Kernel Property

The nested kernel interposes on all modifications of the MMU

1. Access to Page Tables is configured read-only
   - CPU: Base PTR → CR3
   - Memory: PTs
     - Nested Kernel key assumption: can enforce read-only on supervisor code

2. Read-only permissions enforced while the outer kernel executes
   - CPU: CR0 write-protect enable flag
1. Access to Page Tables is configured read-only

   **CPU**: Base PTR \(\rightarrow\) CR3  +  **Memory**: PTs

   Nested Kernel key assumption: can enforce read-only on supervisor code

2. Read-only permissions enforced while the outer kernel executes

   **CPU**: CR0 write-protect enable flag
Initialize read-only

Nested Kernel

```
    nk_update_pte(mapping, pte){
        if (update_pts_to_ptp(val, pte)){
            set_read_only();
        } else {
            do_update();
        }
    }
```

Outer Kernel

nk_update_pte(mapping, pte)
Outer Kernel

nk_update_pte(mapping, pte)
Virtual Privilege Switch Entry Gate

1. Disable interrupts

2. Disable write-protection enforcement

3. Switch to nested kernel stack
Nested Kernel

nk_update_pte(mapping, pte){
    if (update_pts_to_ptp(val, pte)){
        set_read_only();
    }
    do_update();
}
Virtual Privilege Switch Exit Gate

1. Switch to outer kernel stack

2. Enable write-protection enforcement

3. Enable interrupts
Nested Kernel Property

The nested kernel interposes on all modifications of the MMU

1. Access to Page Tables is configured read-only
   - CPU: Base PTR -> CR3
   - Memory: PTs
   - Nested Kernel key assumption: can enforce read-only on supervisor code

2. Read-only permissions enforced while the outer kernel executes
   - CPU: CR0 write-protect enable flag

Diagram:
- Initialize read-only
- Nested Kernel
- Outer Kernel
Enforcing Privilege Separation on x86-64

Code Deprivilege

\[
\text{mov cr3, val } \rightarrow \text{ nk_wr_cr3(val)}
\]

\[
\text{mov cr0, val } \rightarrow \text{ nk_wr_cr0(val)}
\]

Static Code Privilege Separation

Code Verification + Lifetime Kernel Code Integrity

Outer Kernel Execution Integrity
- Caching and Mediation
- Lifetime Integrity
- Non-Privileged Modes
Lifetime Kernel Code Integrity

0. All kernel mappings non-executable

1. If code:
   - verify with scanner
   - configure as executable and read-only

2. User mode code and data: supervisor mode execution prevention
Outer Kernel Execution Integrity

- Guaranteed Mediation
- Return Integrity
- Special Operating Modes
nk_update_ppte:
    disable write-protection
    if valid_update(newvalue, pte):
        *pte = newvalue;  // 0x10
    enable write-protection
    return;
# Nested Kernel Protections

<table>
<thead>
<tr>
<th>CPU Protections</th>
<th>Memory Protections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Page Table Base:</strong> \texttt{mov cr3, val}</td>
<td><strong>Page Tables --- read-only</strong></td>
</tr>
<tr>
<td><strong>Write-protect Enable:</strong> \texttt{mov cr0, val}</td>
<td><strong>Kernel Code --- read-only</strong></td>
</tr>
<tr>
<td><strong>Non-executeable:</strong> \texttt{mov cr4, val}</td>
<td><strong>Kernel Data --- NoExecute</strong></td>
</tr>
<tr>
<td><strong>SMEP:</strong> \texttt{wrmsr(EFER, val)}</td>
<td><strong>User Code + Data --- user-priv</strong></td>
</tr>
</tbody>
</table>
Virtualizing the MMU with the Nested Kernel

Threat Model
Goal: restrict MMU access to nested kernel
- Outer Kernel Under Complete Control of Attacker
- Source Code
- Execution State
- No Control Flow Integrity
- Nested Kernel source is trusted

Isolate the MMU, not defend against general kernel attacks

Nested Kernel Property
The nested kernel intercepts on all modifications of the MMU

Enforcing Privilege Separation on x86-64
Code Deprivilege
- Code Verification
- Lifetime Kernel Code Integrity

The nested kernel isolates the MMU and provides lifetime kernel code integrity
1) What's the problem?

2) Nested Kernel Approach

3) Intra-Kernel Isolation

4) Evaluation

5) Future Work
Nested Kernel Services

memcopy(*obj, *data, size)

Write Mediation

nk_write(*obj,*data,size)
If permitted()

Write-Logging

nk_write(*obj,*data,size)
log_write()
# Intra-kernel memory isolation

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Mitigation</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Injection</td>
<td>Kernel Code Integrity</td>
<td>Read-only</td>
</tr>
<tr>
<td>Function Call Hooking</td>
<td>Syscall Vector Table</td>
<td>Write-Once</td>
</tr>
<tr>
<td>Forensics</td>
<td>Shadow Process List</td>
<td>Log-Write</td>
</tr>
<tr>
<td>Security Monitor</td>
<td>Syscall Recording</td>
<td>Append-Only</td>
</tr>
</tbody>
</table>
Shadow Process List

- Rootkit removes evidence of malicious activities
- Challenge: policy and allocation
- Created a shadow allproc list and force updates to go through nk_log()
- Ensure attacker is observed
Security Monitor

- Attacker cannot remove events
- Guaranteed invocation and isolation of security monitor
- No Virtual Machine Introspection
2) Nested Kernel Approach

3) Intra-Kernel Isolation

4) Evaluation

5) Future Work
How Practical is the Nested Kernel

Kernel Reorganization

Trusted Computing Base

Code Scanner

Microbenchmarks

Macrobenchmark
How Practical is the Nested Kernel

Kernel Reorganization

- 1900 LOC Modified
- 52 Files
- 100 Deleted

DMM - NMI - MMU - Configuration support

Trusted Computing Base

Code Scanner

- 40 implicit instructions
- 38 winner
- 2 writes to cr0
Kernel Reorganization

~1900 LOC Modified

52 Files

~100 Deleted

IOMMU + SMP + NX Configuration support needed
Trusted Computing Base

Nested Kernel Lines of Code

4000 C
800 Assembly
248 Python Scanner

TCB for Nested Kernel and MMU Isolation

FreeBSD 9.0: 7.9 MB
PerspicuOS: 34 KB

MMU TCB FreeBSD 9.0 / PerspicuOS: 232x
Nested Kernel Lines of Code

4000 C

800 Assembly

248 Python Scanner
TCB for Nested Kernel and MMU Isolation

FreeBSD 9.0: 7.9 MB
PerspicuOS: 34 KB

MMU TCB FreeBSD 9.0 / PerspicuOS: 232x
Code Scanner

40 implicit instructions

2 writes to cr0

38 wrmsr
Microbenchmarks

![LMBENCH Syscall Overheads](image)

- **PerspicuOS**
- **Syscall Logging**
- **System Call Table**
- **Shadow Process List**
Macrobenchmark

Apache HTTPD

Apache Bench Bandwidth Relative to FreeBSD 9.0

Kernel Make Overhead

SSHD Server Bandwidth

Each connection forks a new process and launches the command.
Apache HTTPD

Apache Bench Bandwidth Relative to FreeBSD 9.0

File Size (KB)

Bandwidth Relative to FreeBSD 9.0

- PerspicuOS
- Syscall Logging
- System Call Table
- Shadow Process List
SSH Server Bandwidth

Each connection forks a new process and mmaps data.
3) Intra-Kernel Isolation

4) Evaluation

5) Future Work
Directions for the Nested Kernel
Completing the Implementation

To Finish:

- NX Policies
- SMP
- Special operating modes (e.g., SMM)
- IOMMU Support
- MMU Operation Batching
- Port to FreeBSD Current and Refactor

To Add:

- Hardware-assisted virtualization (e.g. VT-x)
- Dynamic rewriting for maintenance
Future Research

• **End-to-end** intra-kernel security solutions

• **Compelling** memory isolation use cases (e.g., device drivers, file system, crypto)

• **De-privileging** to virtualize other hardware (e.g., EFLAGS)

• Nested Kernel **VMMs**

• **Formal verification** of Nested Kernel
Key Takeaways

- Retrofit the nested kernel architecture in FreeBSD 9.0

- Isolate MMU at a single hardware privilege level by virtualizing supervisor mode

- Intra-kernel memory protection services

- Lifetime kernel code integrity in FreeBSD 9.0