Fast Execute-Only Memory for Embedded Systems

Zhuojia Shen, Komail Dharsee, and John Criswell
University of Rochester
Code Disclosure Attacks

Dangerous
Code Disclosure Attacks

Dangerous

- Steal intellectual property (e.g., proprietary code)
Code Disclosure Attacks

Dangerous
- Steal intellectual property (e.g., proprietary code)
- Find gadgets for code reuse attacks (e.g., ROP)
Code Disclosure Attacks

Dangerous

- Steal intellectual property (e.g., proprietary code)
- Find gadgets for code reuse attacks (e.g., ROP)

Even worse on embedded systems
Code Disclosure Attacks

Dangerous

- Steal intellectual property (e.g., proprietary code)
- Find gadgets for code reuse attacks (e.g., ROP)

Even worse on embedded systems

- Internet connectivity due to IoT
Code Disclosure Attacks

Dangerous

- Steal intellectual property (e.g., proprietary code)
- Find gadgets for code reuse attacks (e.g., ROP)

Even worse on embedded systems

- Internet connectivity due to IoT
- Lack of protection
Execute-Only Memory (XOM)

Memory that only allows instruction fetching
- No reading or writing allowed
- Best fit for code disclosure defenses
Execute-Only Memory (XOM)

Memory that only allows instruction fetching
- No reading or writing allowed
- Best fit for code disclosure defenses

Not natively available on ARM embedded systems
Execute-Only Memory (XOM)

Memory that only allows instruction fetching
- No reading or writing allowed
- Best fit for code disclosure defenses

Not natively available on ARM embedded systems
- uXOM\textsuperscript{[1]}: significant performance and code size overhead

\textsuperscript{[1]} D. Kwon et al. “uXOM: Efficient eXecute-Only Memory on ARM Cortex-M”. In: USENIX Security ’19.
Execute-Only Memory (XOM)

Memory that only allows instruction fetching
- No reading or writing allowed
- Best fit for code disclosure defenses

Not natively available on ARM embedded systems
- uXOM\cite{1}: significant performance and code size overhead
- PCROP\cite{2}: only support flash memory on STMicroelectronics devices

\begin{flushright}
\footnotesize
\cite{1} D. Kwon et al. “uXOM: Efficient eXecute-Only Memory on ARM Cortex-M”. In: USENIX Security '19.
\end{flushright}
Execute-Only Memory (XOM)

Memory that only allows instruction fetching
- No reading or writing allowed
- Best fit for code disclosure defenses

Not natively available on ARM embedded systems
- uXOM\cite{1}: significant performance and code size overhead
- PCROP\cite{2}: only support flash memory on STMicroelectronics devices
- Others: require an MMU

\cite{1} D. Kwon et al. “uXOM: Efficient eXecute-Only Memory on ARM Cortex-M”. In: USENIX Security ’19.
We present *PicoXOM*: a novel XOM solution for ARM embedded systems

- Uses ARM’s debug support to achieve execute-only memory
- Works on both ARMv7-M and ARMv8-M architectures
- Efficient: 0.33% performance and 5.89% code size overhead
Outline

I  Design

II  Implementation

III  Evaluation

IV  Summary
Background on ARMv7-M and ARMv8-M

### Execution modes

- Privileged and unprivileged
- Software by default executes in privileged mode
Background on ARMv7-M and ARMv8-M

Execution modes

- Privileged and unprivileged
- Software by default executes in privileged mode

Memory Protection Unit (MPU)

- Region-based access control on physical address space
- No XO permission
# Background on ARMv7-M and ARMv8-M

## Execution modes
- Privileged and unprivileged
- Software by default executes in privileged mode

## Memory Protection Unit (MPU)
- Region-based access control on physical address space
- No XO permission

## Data Watchpoint and Tracing (DWT)
- Part of ARM debug support
- Groups of registers called *DWT Comparators*
- Able to monitor accesses to an *address range*
  - Trap on matched access
  - Otherwise no effect on performance
Enforcing XOM

$W \oplus X$ by MPU

- Code region: RX
- Read-only data: RO
- Other regions: RW

![Diagram of memory regions and comparators]
Enforcing XOM

\( W \oplus X \) by MPU
- Code region: RX
- Read-only data: RO
- Other regions: RW

\( R \oplus X \) by DWT
- Configure a DWT comparator to watch over code region for read accesses
- Illegal code reads will trap
Enforcing XOM

Prevent neutralization

- MPU and DWT registers are memory-mapped
Enforcing XOM

Prevent neutralization

- MPU and DWT registers are memory-mapped
- Two approaches
  - Run in unprivileged mode
Enforcing XOM

Prevent neutralization

- MPU and DWT registers are memory-mapped
- Two approaches
  - Run in unprivileged mode
  - Write-protection by DWT
Constant Island Removal

Constant data embedded in code region (a.k.a. “constant island”) no longer readable
Constant Island Removal

Constant data embedded in code region (a.k.a. “constant island”) no longer readable

- Load constant

```assembly

ldr r0, =L
...
L: .word 0x12345678
```

Jump-table jump (TBB and TBH)
```
tbb [ pc , r2 ]
```
```
.L0:
.L1 - L0 ) / 2
.L2 - L0 ) / 2
.L3 - L0 ) / 2
...
L1 :
L2 :
L3 :
```
```
adr.w r1 , =L0
add.w r1 , r1 , r2 , lsl #2
mov pc , r1;
```
```
b.w L1
b.w L2
b.w L3
```

Transform the two instructions and remove “constant islands”
Constant Island Removal

Constant data embedded in code region (a.k.a. “constant island”) no longer readable

- Load constant

```
  ldr  r0, =L
  ...  
  L:  .word 0x12345678
```

- Jump-table jump (TBB and TBH)

```
  tbb  [pc, r2]
  L0:  .byte (L1 - L0) / 2
       .byte (L2 - L0) / 2
       .byte (L3 - L0) / 2
  ...  
  L1:  ...
  L2:  ...
  L3:  ...
```
Constant Island Removal

Constant data embedded in code region (a.k.a. “constant island”) no longer readable

- Load constant

```
ldr  r0, =L  
... 
L:  .word 0x12345678
```

```
movw r0, #0x5678  
movt r0, #0x1234  
... 
```

- Jump-table jump (TBB and TBH)

```
tbb  [pc, r2] 
L0:  .byte (L1 - L0) / 2  
     .byte (L2 - L0) / 2  
     .byte (L3 - L0) / 2  
     ... 
L1:  ...  
L2:  ...  
L3:  ...  
... 
```

```
adr.w r1, =L0  
add.w r1, r1, r2, lsl #2  
mov  pc, r1 ; indirect jump  
b.w  L1  
b.w  L2  
b.w  L3  
... 
```

Transform the two instructions and remove “constant islands”
PicoXOM Workflow

Application Source Code

MPU Configuration Code

DWT Configuration Code

LLVM IR

Constant Island Removal

PicoXOM Binary

PicoXOM Run-time

PicoXOM Compiler
Outline

I  Design

II  Implementation

III  Evaluation

IV  Summary
Implementation

Basics

- Prototype on ARMv7-M
- MPU and DWT configuration code in C (177 SLoC)
- Constant Island Removal as an LLVM IR pass (88 SLoC)

One limitation

ARMv7-M only supports up to 4 DWT comparators
Each DWT comparator can only monitor up to 32 KB address range
Each address range must be sized and aligned by a power-of-2
Maximum code size of 128 KB
No such limit on ARMv8-M
Implementation

Basics

- Prototype on ARMv7-M
- MPU and DWT configuration code in C (177 SLoC)
- Constant Island Removal as an LLVM IR pass (88 SLoC)

One limitation

- ARMv7-M only supports up to 4 DWT comparators
- Each DWT comparator can only monitor up to 32 KB address range
- Each address range must be sized and aligned by a power-of-2
- Maximum code size of 128 KB
Implementation

Basics

- Prototype on ARMv7-M
- MPU and DWT configuration code in C (177 SLoC)
- Constant Island Removal as an LLVM IR pass (88 SLoC)

One limitation

- ARMv7-M only supports up to 4 DWT comparators
- Each DWT comparator can only monitor up to 32 KB address range
- Each address range must be sized and aligned by a power-of-2
- Maximum code size of 128 KB

No such limit on ARMv8-M
Outline

I  Design

II  Implementation

III  Evaluation

IV  Summary
Experimental Setup

Device specification

- STM32F469 Discovery board
- ARM Cortex-M4 processor @ 180 MHz
- 2 MB flash memory, 384 KB SRAM, and 16 MB SDRAM
- LCD screen and microSD card slot
Experimental Setup

Device specification

- STM32F469 Discovery board
- ARM Cortex-M4 processor @ 180 MHz
- 2 MB flash memory, 384 KB SRAM, and 16 MB SDRAM
- LCD screen and microSD card slot

Benchmarks and applications

- 42 (out of 80) programs in BEEBS
- CoreMark-Pro
- 4 applications from manufacturer
- PinLock
Experimental Setup

Device specification

- STM32F469 Discovery board
- ARM Cortex-M4 processor @ 180 MHz
- 2 MB flash memory, 384 KB SRAM, and 16 MB SDRAM
- LCD screen and microSD card slot

Benchmarks and applications

- 42 (out of 80) programs in BEEBS
- CoreMark-Pro
- 4 applications from manufacturer
- PinLock

Settings

- Baseline: LLVM/Clang 10.0 -Os
- PicoXOM: Baseline + MPU + DWT + Constant Island Removal
Performance Overhead on BEEBS

<table>
<thead>
<tr>
<th>33 BEEBS programs in common</th>
<th>uXOM</th>
<th>PicoXOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>~0%</td>
<td>-1.27%</td>
</tr>
<tr>
<td>Max</td>
<td>~23%</td>
<td>7.48%</td>
</tr>
<tr>
<td>Geo. mean</td>
<td>7.3%</td>
<td><strong>0.55%</strong></td>
</tr>
</tbody>
</table>
Performance Overhead on CoreMark-Pro

<table>
<thead>
<tr>
<th>CoreMark-Pro</th>
<th>PicoXOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>-0.82%</td>
</tr>
<tr>
<td>Max</td>
<td>0.17%</td>
</tr>
<tr>
<td>Geo. mean</td>
<td>-0.11%</td>
</tr>
</tbody>
</table>
Performance Overhead on Applications

<table>
<thead>
<tr>
<th>Applications</th>
<th>PicoXOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>−0.11%</td>
</tr>
<tr>
<td>Max</td>
<td>0.22%</td>
</tr>
<tr>
<td>Geo. mean</td>
<td>0.02%</td>
</tr>
</tbody>
</table>

Normalized Execution Time

FatFs-RAM FatFs-uSD LCD-Animation LCD-uSD PinLock

V ariances due to access to slow peripherals
Performance Overhead on Applications

The graph shows the normalized execution time for different applications under PicoXOM. The applications include FatFs-RAM, FatFs-uSD, LCD-Animation, LCD-uSD, and PinLock. The normalized execution times range from 0.99 to 1.01.

<table>
<thead>
<tr>
<th>Applications</th>
<th>PicoXOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>-0.11%</td>
</tr>
<tr>
<td>Max</td>
<td>0.22%</td>
</tr>
<tr>
<td>Geo. mean</td>
<td>0.02%</td>
</tr>
</tbody>
</table>

Variance due to access to slow peripherals.
Code Size Overhead

![Bar chart showing code size overhead for different programs with baseline and PicoXOM results.]

### 33 BEEBS programs in common

<table>
<thead>
<tr>
<th></th>
<th>uXOM</th>
<th>PicoXOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>~0%</td>
<td>3.29%</td>
</tr>
<tr>
<td>Max</td>
<td>~20%</td>
<td>6.54%</td>
</tr>
<tr>
<td>Geo. mean</td>
<td>15.7%</td>
<td>6.10%</td>
</tr>
</tbody>
</table>

### Overall

<table>
<thead>
<tr>
<th></th>
<th>PicoXOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>3.29%</td>
</tr>
<tr>
<td>Max</td>
<td>7.93%</td>
</tr>
<tr>
<td>Geo. mean</td>
<td>5.89%</td>
</tr>
</tbody>
</table>
Outline

I  Design

II  Implementation

III  Evaluation

IV  Summary
PicoXOM:

- Novel XOM realization for ARM embedded systems
- First use of ARM debug features for enforcing security
- Practically *no* performance overhead and low code size overhead
Summary

PicoXOM:

- Novel XOM realization for ARM embedded systems
- First use of ARM debug features for enforcing security
- Practically *no* performance overhead and low code size overhead

Questions?