FlexFilt: Towards Flexible Instruction Filtering for Security

Leila Delshadtehrani, Sadullah Canakci, William Blair, Manuel Egele, and Ajay Joshi

delshad@bu.edu

Boston University

December 9, 2021
Motivation

- How to limit the effects of bugs and security vulnerabilities?
  - Isolation-based mechanisms

- How to guarantee the integrity of isolation-based mechanisms?
  - Prevent the execution of various unsafe instructions in untrusted parts of the code (either in user space or kernel space)
  - Potential effects of unsafe instructions
    - Modify access permissions, disable protections, gain higher privilege, etc.
Motivational Example: Intel MPK

Memory Protection Keys (MPK)

- Per page protection keys (pkeys)
  - PKRU: a single 32-bit register storing the permission bits of each pkey
  - WRPKRU: a new user-space instruction to write into PKRU

<table>
<thead>
<tr>
<th>VPage#</th>
<th>PPage#</th>
<th>Page-Table Perm</th>
<th>Pkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>1223</td>
<td>10</td>
<td>0001</td>
</tr>
<tr>
<td>110</td>
<td>2089</td>
<td>11</td>
<td>1111</td>
</tr>
<tr>
<td>87</td>
<td>760</td>
<td>11</td>
<td>0001</td>
</tr>
</tbody>
</table>

DTLB

PPage#

<table>
<thead>
<tr>
<th>RW</th>
<th>Pkey</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Vpage# 110
Effective RW Perm: 10
Motivational Example: Intel MPK

Memory Protection Keys (MPK)
- Per page protection keys (pkeys)
  - PKRU: a single 32-bit register storing the permission bits of each pkey
  - WRPKRU: a new user-space instruction to write into PKRU

WRPKRU Security Challenge
- An untrusted component can gain access permission to any protection domain by simply writing into PKRU
- Previous solutions
  - Binary scanning and binary rewriting
    - Hodor [Hedayati, ATC’19] and ERIM [Vahldiek-Oberwagner, Security’19]
  - Hardware-assisted call-gates
    - Donky [Schrammel, Security’20]
### Instruction Filtering in Prior Works

<table>
<thead>
<tr>
<th>x86</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>- WRPKRU instruction</td>
<td>- MSR [Zhou, Security’20]</td>
</tr>
<tr>
<td>- Extended instructions,</td>
<td></td>
</tr>
<tr>
<td>e.g., SMOV [Frassetto, Security’18]</td>
<td>: Binary scanning</td>
</tr>
<tr>
<td>: CFI</td>
<td></td>
</tr>
<tr>
<td>- MOV CR3 [Wu, HPCA’18], [Gu, ATC’20]</td>
<td>- LDC, MCR [Azab, CCS’14], [Azab, NDSS’16]</td>
</tr>
<tr>
<td>: Binary scanning and binary rewriting</td>
<td>: Binary scanning</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RISC-V</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Extended instructions,</td>
</tr>
<tr>
<td>e.g., WRPKR [Delshadtehrani, DATE’21]</td>
</tr>
<tr>
<td>: Dedicated hardware</td>
</tr>
</tbody>
</table>
Prior Works: Challenges and Limitations

**Challenges**
- Implicit occurrences of target instructions [Hedayati, ATC’19], [Vahldiek-Oberwagner, Security’19]
- Just-In-Time (JIT) compiled code [Schrammel, Security’20]

**Limitations**
- Limited to filtering the execution of fixed target instructions [Hedayati, ATC’19], [Vahldiek-Oberwagner, Security’19], etc.
- High performance overhead of dynamic binary rewriting tools [Bauman, NDSS’18], [Gorgovan, TACO’16]
FlexFilt: Overview

Goal
- Provide a generalized solution for filtering target instructions
  - Flexible
  - Efficient
  - Fine-grained

Target instructions
- Unsafe instructions whose execution should be prevented in untrusted parts of the code
An efficient and flexible hardware-assisted capability for runtime filtering of target instructions at page granularity

- Creates instruction domains
- Prevents the execution of configured target instructions at page granularity in each domain
- Capable of filtering privileged instructions
Follow the common threat model in prior work

- Untrusted parts of the code might contain vulnerabilities that an adversary can exploit to inject or reuse arbitrary instructions including the target instructions

- Safe occurrences of target instructions in trusted parts of the code are surrounded by call gates or trampoline

- All hardware components are trusted

- OS is partially trusted
Hardware Overview

Instruction Protection Domains

- Up to 16 instruction domains

Flexible Filters

- Four shared configurable instruction filters
  - Each instruction domain applies a combination of the flexible filters
  - Each instruction filter can be configured to filter various target instructions
    - A bit-granular matching mechanism on the instruction (e.g., match the opcode)

Runtime instruction: 0x03776263 (bltu a4,s7,112aa)
Hardware Design

- **Modified MMU**
- **Instruction Protection Register (IPR)** to store the ipkey information
- Cause an exception to prevent the execution of unsafe instructions
- Less than 1% area overhead according to FPGA resource utilization
Software Overview

**OS Support**
- Support for instruction protection keys
  - Built on top of the existing support for memory protection keys
- Per process OS support
  - FlexFilt information maintained during context switches

**Software Support**
- Software API leveraging RISC-V custom instruction
- Proof of concept by leveraging LD_PRELOAD
Motivation

- Binary rewriting
  - Filtering target instructions in dynamically generated code is challenging

- JIT code
  - A popular use-case of dynamically generated code
Motivation

- **Binary rewriting**
  - Filtering target instructions in dynamically generated code is challenging

- **JIT code**
  - A popular use-case of dynamically generated code

---

**V8 JIT Compilation Experiment**

- **Alexa top10 websites**
  - Built Chromium with `v8_enable_disassembler=true`
  - Measured the total number of generated bytes (`–js-flags=“–print-bytecode”`)

<table>
<thead>
<tr>
<th>Website</th>
<th>Executable bytes generated when loading the frontpage</th>
<th>Executable bytes generated per second while browsing the page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google.com</td>
<td>0</td>
<td>3,458</td>
</tr>
<tr>
<td>Youtube.com</td>
<td>266,798</td>
<td>2,620</td>
</tr>
<tr>
<td>Tmall.com</td>
<td>366,003</td>
<td>15,323</td>
</tr>
<tr>
<td>Baidu.com</td>
<td>0</td>
<td>1,532</td>
</tr>
<tr>
<td>Qq.com</td>
<td>159,565</td>
<td>2,043</td>
</tr>
<tr>
<td>Sohu.com</td>
<td>34,096</td>
<td>2,014</td>
</tr>
<tr>
<td>Facebook.com</td>
<td>20,938</td>
<td>9,712</td>
</tr>
<tr>
<td>Taobao.com</td>
<td>220,299</td>
<td>15,454</td>
</tr>
<tr>
<td>Amazon.com</td>
<td>92,442</td>
<td>3,098</td>
</tr>
<tr>
<td>360.cn</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>Geometric mean</td>
<td>3,432</td>
<td>3,258</td>
</tr>
</tbody>
</table>
**Motivation**

- **Binary rewriting**
  - Filtering target instructions in dynamically generated code is challenging

- **JIT code**
  - A popular use-case of dynamically generated code

---

**V8 JIT Compilation Experiment**

- **Alexa top10 websites**
  - Built Chromium with `v8_enable_disassembler=true`
  - Measured the total number of generated bytes (`--js-flags="--print-bytecode"`)  

<table>
<thead>
<tr>
<th>Website</th>
<th>Executable bytes generated when loading the frontpage</th>
<th>Executable bytes generated per second while browsing the page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google.com</td>
<td>0</td>
<td>3,458</td>
</tr>
<tr>
<td>Youtube.com</td>
<td>266,798</td>
<td>2,620</td>
</tr>
<tr>
<td>Tmall.com</td>
<td>366,003</td>
<td>15,323</td>
</tr>
<tr>
<td>Baidu.com</td>
<td>0</td>
<td>1,532</td>
</tr>
<tr>
<td>Qq.com</td>
<td>159,565</td>
<td>2,043</td>
</tr>
<tr>
<td>Sohu.com</td>
<td>34,096</td>
<td>2,014</td>
</tr>
<tr>
<td>Facebook.com</td>
<td>20,938</td>
<td>9,712</td>
</tr>
<tr>
<td>Taobao.com</td>
<td>220,299</td>
<td>15,454</td>
</tr>
<tr>
<td>Amazon.com</td>
<td>92,442</td>
<td>3,098</td>
</tr>
<tr>
<td>360.cn</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>Geometric mean</td>
<td>3,432</td>
<td>3,258</td>
</tr>
</tbody>
</table>

---

FlexFilt prevents the execution of unsafe instructions without the need for binary scanning and binary rewriting
Implementation and Evaluation Framework

**Implementation**
- FlexFilt written in Chisel HDL
  - Implemented on the in-order RISC-V Rocket core
- Linux kernel v4.15
- RISC-V gnu toolchain for cross-compilation

**Evaluation**
- Prototyped on Xilinx Zynq Zedboard
  - Rocket core + FlexFilt
- Open-source coming soon: https://github.com/bu-icsg/FlexFilt
**Evaluation Results**

**Functional Verification**

- **User-space target instruction**
  - Prevented the execution of an untrusted instruction in an untrusted domain
  - Leveraged a buffer overflow vulnerability in a simple program to inject a WRPKR instruction and prevent its execution in an untrusted domain

- **Kernel-level target instruction**
  - Proof of concept evaluation
    - Configured FlexFilt in BBL to limit the execution of our custom instructions
### Evaluation Results

#### Functional Verification

- **User-space target instruction**
  - Prevented the execution of an untrusted instruction in an untrusted domain
  - Leveraged a buffer overflow vulnerability in a simple program to inject a WRPKR instruction and prevent its execution in an untrusted domain

- **Kernel-level target instruction**
  - Proof of concept evaluation
    - Configured FlexFilt in BBL to limit the execution of our custom instructions

#### Performance Evaluation

- **Microbenchmarks**
  - Regardless of the number of activated configured filters, FlexFilt’s performance overhead remains the same

- **Macrobenchmarks**
  - Negligible performance overhead for SPEC 2000 and SPEC 2006 benchmarks (less than 0.1%)
**Conclusion**

- Guarantees the integrity of isolation-based mechanisms efficiently without binary scanning and binary rewriting.
- Filters configured instructions at page granularity.

**Artifact Evaluated**

https://github.com/bu-icsg/FlexFilt

Thanks! Reach me at delshad@bu.edu for questions.