FlexFilt: Towards Flexible Instruction Filtering for Security

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

delshad@bu.edu

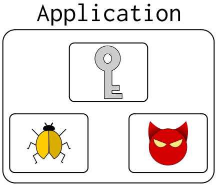
Boston University

December 9, 2021

Runtime Instruction Filtering

Motivation

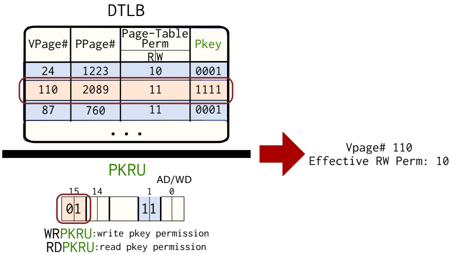
- How to limit the effects of bugs and security vulnerabilities?
  - Isolation-based mechanisms
- How to guarantee the integrity of isolation-based mechanisms?
  - Prevent the execution of various **unsafe** instructions in untrusted parts of the code (either in user space or kernel space)
  - Potential 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



DTLB

VPage#	PPage#	Page-Table Perm R W	Pkey
24	1223	10	0001
110	2089	11	1111
87	760	11	0001
. . .			

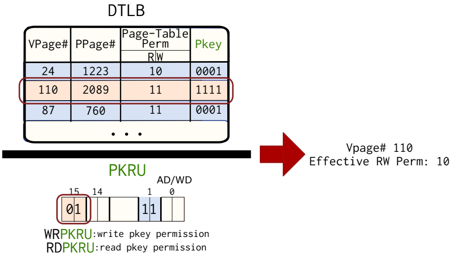
WRPKRU:write pkey permission
RDPKRU:read pkey permission

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



DTLB

VPage#	PPage#	Page-Table Perm R|W	Pkey
24	1223	10	0001
110	2089	11	1111
87	760	11	0001
. . .			

Vpage# 110
Effective RW Perm: 10

WRPKRU:write pkey permission
RDPKRU:read pkey permission

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

x86

- WRPKRU instruction

- Extended instructions, e.g., SMOV [Frassetto, Security'18]
: CFI

- MOV CR3 [Wu, HPCA'18], [Gu, ATC'20]
: Binary scanning and binary rewriting

ARM

- MSR [Zhou, Security'20]
: Binary scanning

- LDC, MCR [Azab, CCS'14], [Azab, NDSS'16]
: Binary scanning

RISC-V

- Extended instructions, e.g., WRPKR [Delshadtehrani, DATE'21]
: Dedicated hardware

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

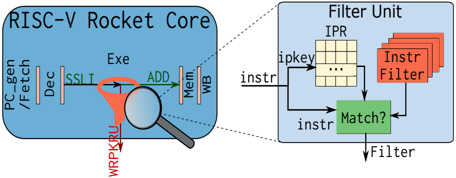
- Provide a generalized solution for filtering **target** instructions
    - Flexible
    - Efficient
    - Fine-grained

Target instructions

- Unsafe instructions whose execution should be prevent in untrusted parts of the code

FlexFilt

- 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



Threat Model

- 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

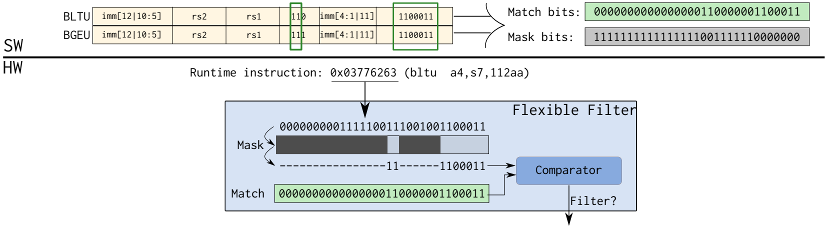
Hardware Overview

Instruction Protection Domains

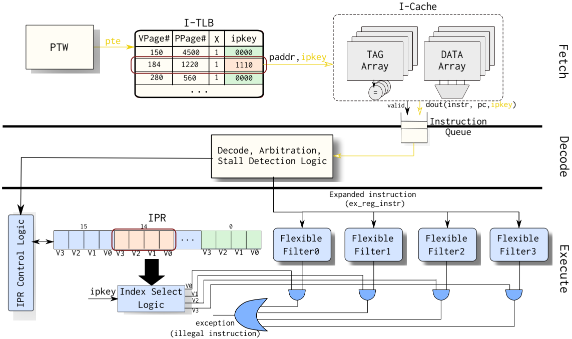
- Up to 16 instruction domains

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)



Hardware Design



VPage#	PPage#	X	ipkey
150	4500	1	0000
184	1220	1	1110
280	560	1	0000
...			

- 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`

Case Study

Motivation

- Binary rewriting
    - Filtering target instructions in dynamically generated code is challenging
- JIT code
    - A popular use-case of dynamically generated code

Case Study

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”)

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

Case Study

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”)

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

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

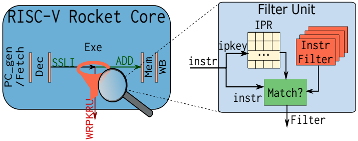
- User-space target instruction
  - Prevented the execution of an untrusted instruction in an untrusted domain
  - Leveraged a buffer overflow vulnerability in a simple program to inject a WRPKR instruction and prevent its execution in an untrusted domain
- Kernel-level target instruction
  - Proof of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