Pointing in the Right Direction – Securing Memory Accesses in a Faulty World

Robert Schilling[1,2], Mario Werner[1], Pascal Nasahl[1], Stefan Mangard[1]

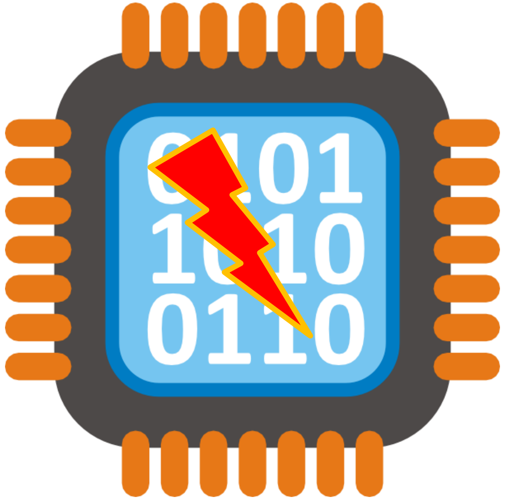
[1]Graz University of Technology, [2]Know-Center GmbH

December 06th, 2018

Our Faulty World

Clock Glitch

Voltage
Glitch



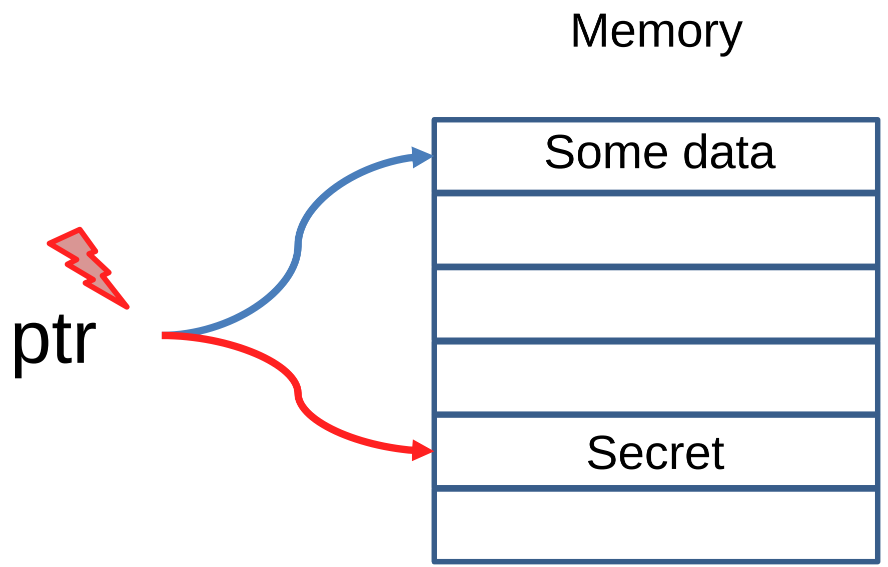
Laser

Motivation

- Fault attacks modify code and data
    - Use Control-Flow Integrity to restrict the control-flow
    - Data encoding to protect data and arithmetic
- **No protection for memory accesses**
- Memory accesses are critical
    - There is a lot of critical information in the memory
    - **How** to ensure we read from the correct location?

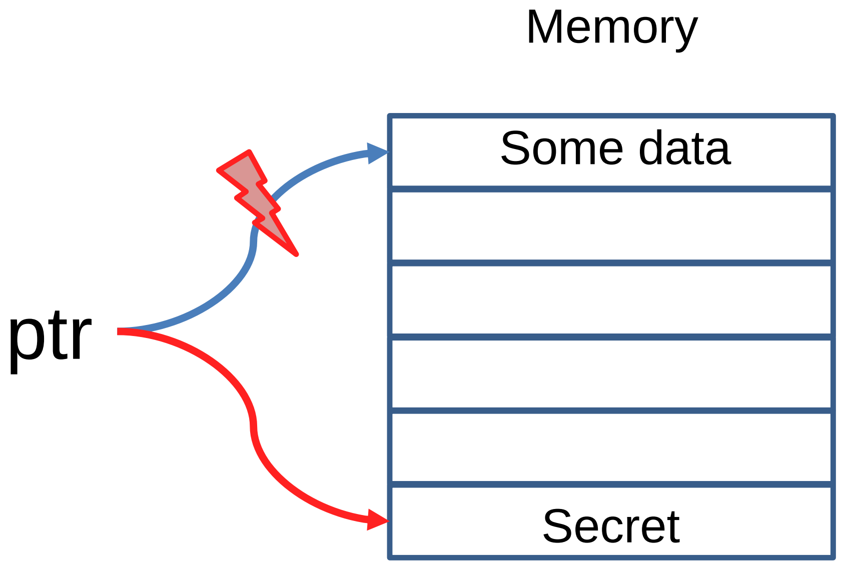
Attack Vector for Memory Accesses

- Faulted pointer redirects the memory access



Attack Vector for Memory Accesses

- Faulted pointer redirects the memory access
- Faulting the memory access itself leads to a wrong access



Pointer Protection with Residue Codes

- Pointers are ubiquitous
  - Every memory access uses some kind of pointer
- Pointers are unprotected
  - Faults can manipulate the pointer to point to a different memory location
- Pointers require a **redundant** encoding
  - We use a multi-residue code to protect pointers

A Primer to Multi-Residue Codes

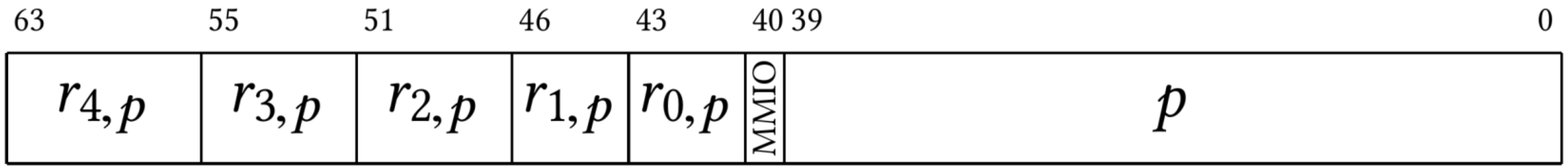
- Arithmetic code with support for addition/subtraction
- Separable code → Tuple representation
- $p_r = (p \mid r_{p,1} \mid \ldots \mid r_{p,n})$ with $r_{p,i} = p \bmod m_i$ and $M = \{m_1, \ldots, m_n\}$
- $z_r = x_r + y_r$
  $= (x + y \mid \forall\, i\colon (r_{x,i} + r_{y,i}) \bmod m_i)$
- Used to perform pointer arithmetic

Pointer Protection with Residue Codes

- Use multi-residue code to protect the pointer
  - Gives direct access to the functional value $\rightarrow$ no expensive decoding required
  - Supports pointer arithmetic
- But where to store the redundancy information?
  - Parallel register file
  - A pair of regular registers
  - Reduce address space and store it in the pointer

Pointer Layout

- Target a 64-bit platform
- Use a multi-reside code with five residues and a modulus size of 23-bit with 5-bit Hamming distance
- Resulting pointer layout:



Pointer Operations

- Software approach not practicable
- Instruction set extension for pointer manipulation
  - **radd/rsub** – Add/subtract two residue encoded values
  - **raddi** – Add an immediate to a residue encoded value
  - **renc** – Encode a value to the residue domain
  - **rdec** – Decode and remove the redundancy information

Secure Memory Accesses

- Pointers are protected but memory access still can be redirected
- Establish a link between the redundant address and redundant data
- Perform a linking overlay on top of encoded data
- Unlinking operation only successful when using the correct pointer and correct memory access

  → Translate addressing errors to data errors

Linking Approach

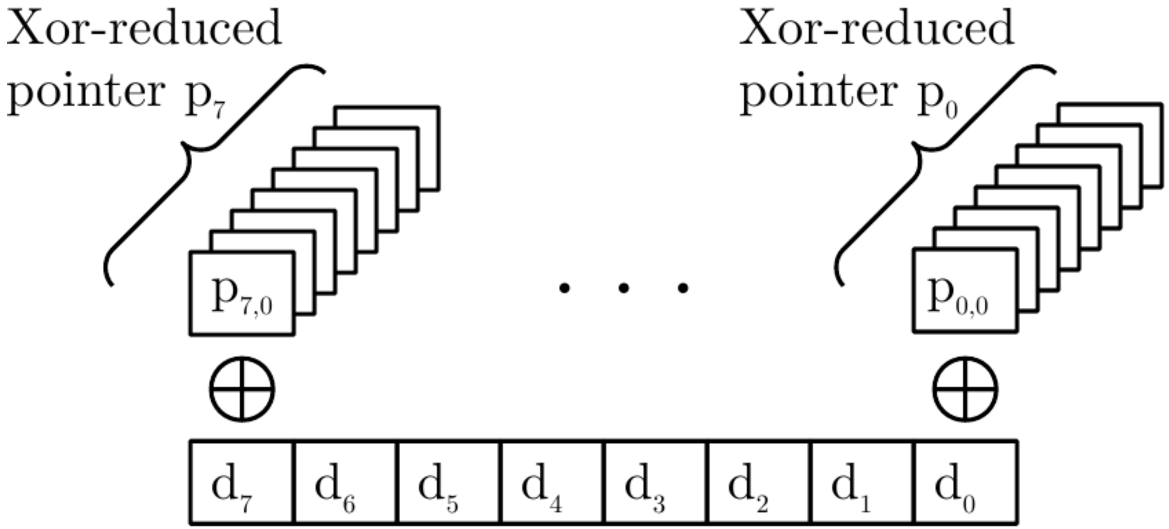
- Write memory in the form $mem[p] = l_p(D_{Reg})$
- Inverse to read data back $D_{Reg} = l_p^{-1}(mem[p])$
- Xor operation → chosen for low-overhead
  - $mem[p] = p \oplus D_{Reg}, \qquad D_{Reg} = p \oplus mem[p]$
  - **Problems** with granularity

Linking Granularity

- Coarse grain link does not add enough diffusion
  - Close bytes (8 bytes stride on a 64-bit system) likely have the same address pad
- Misaligned data accesses with arbitrary size not supported, e.g. for $memcpy$
- Use a byte-wise linking granularity

Byte-Wise Data Linking

- Compute the xor-reduced address pad for each byte address
- Better diffusion and support for misaligned accesses


Xor-reduced pointer p_7
Xor-reduced pointer p_0
$p_{7,0}$
$p_{0,0}$
$\oplus$
d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

Instruction Set Extensions for Memory Accesses

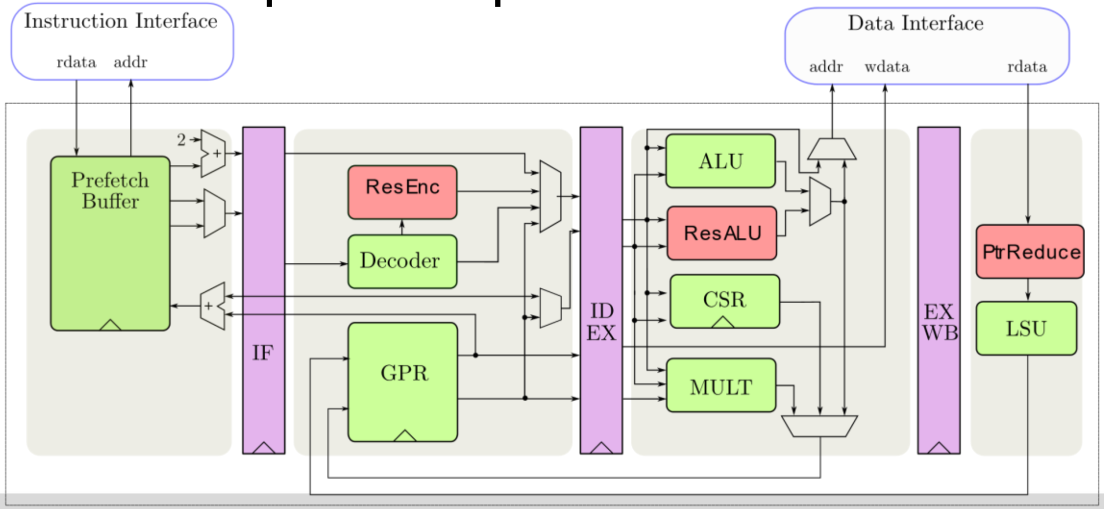
- rs$x$ck
    - Stores one memory element of granularity $x \in \{b, h, w, d\}$ using a protected pointer and performs memory linking
- rl$x$ck
    - Loads one memory element of granularity $x \in \{b, h, w, d\}$ using a protected pointer and performs memory unlinking

LLVM Compiler Prototype

- Transformation performed in the backend → target dependent
- Identify address generation in the SelectionDAG, encode, and propagate residue information down to memory accesses
- Linker fills encoded relocations
- Supports compilation of large code bases

RISC-V Hardware Architecture

- 32-bit RISC-V core RI5CY from PULP SoC extended to 64-bit
- Register file, datapath, load-and-store unit extended
- Dedicated residue ALU for pointer operations



Evaluation Setting

- FPGA prototype based on PULP with 5% overhead on Xilinx Artix-7 FPGA
- ISA extension residue arithmetic and linked memory accesses
- Transformed all data pointers, protected all pointer arithmetic, replaced all memory accesses
- Evaluated code overhead and runtime in cycles

Evaluation Results

Benchmark	Code Overhead		Runtime Overhead	
	Baseline [kb]	Overhead [%]	Baseline [kCycles]	Overhead [%]

Evaluation Results

Benchmark	Code Overhead		Runtime Overhead	
	Baseline [kb]	Overhead [%]	Baseline [kCycles]	Overhead [%]
fir				

Evaluation Results

Benchmark	Code Overhead		Runtime Overhead	
	Baseline [kb]	Overhead [%]	Baseline [kCycles]	Overhead [%]
fir	4.26			

Evaluation Results

Benchmark	Code Overhead		Runtime Overhead	
	Baseline [kb]	Overhead [%]	Baseline [kCycles]	Overhead [%]
fir	4.26	8.54		

Evaluation Results

Benchmark	Code Overhead		Runtime Overhead	
	Baseline [kb]	Overhead [%]	Baseline [kCycles]	Overhead [%]
fir	4.26	8.54	39.22	

Evaluation Results

Benchmark	Code Overhead		Runtime Overhead	
	Baseline [kb]	Overhead [%]	Baseline [kCycles]	Overhead [%]
fir	4.26	8.54	39.22	6.35

Evaluation Results

Benchmark	Code Overhead		Runtime Overhead	
	Baseline [kb]	Overhead [%]	Baseline [kCycles]	Overhead [%]
fir	4.26	8.54	39.22	6.35
fft	6.52	6.57	58.01	4.65
keccak	4.79	10.11	255.55	11.31
ipm	4.84	12.81	10.80	3.94
aes_cbc	7.25	8.77	60.91	9.10
conv2d	3.26	13.11	5.92	2.7

Evaluation Results

Benchmark	Code Overhead		Runtime Overhead	
	Baseline [kb]	Overhead [%]	Baseline [kCycles]	Overhead [%]
fir	4.26	8.54	39.22	6.35
fft	6.52	6.57	58.01	4.65
keccak	4.79	10.11	255.55	11.31
ipm	4.84	12.81	10.80	3.94
aes_cbc	7.25	8.77	60.91	9.10
conv2d	3.26	13.11	5.92	2.7
Average		**9.99**		**6.34**

Improvements

- Not all pointer arithmetic is supported
- Unsupported operations are decoded, performed in the unprotected domain, and then reencoded
- Compiler has early support for RISC-V
    - More optimized compiler increases code quality and reduces code size

Conclusion

- Protect **all** data pointers and memory accesses
- Encode pointers with a multi-residue code supporting pointer arithmetic
    - Store redundancy in the upper bits of the pointer
- Perform memory linking on byte-wise granularity
    - Translate addressing errors to data errors
- Integrate concept to RISC-V FPGA prototype and LLVM

S C I E N C E ▪ P A S S I O N ▪ T E C H N O L O G Y

Pointing in the Right Direction – Securing Memory Accesses in a Faulty World

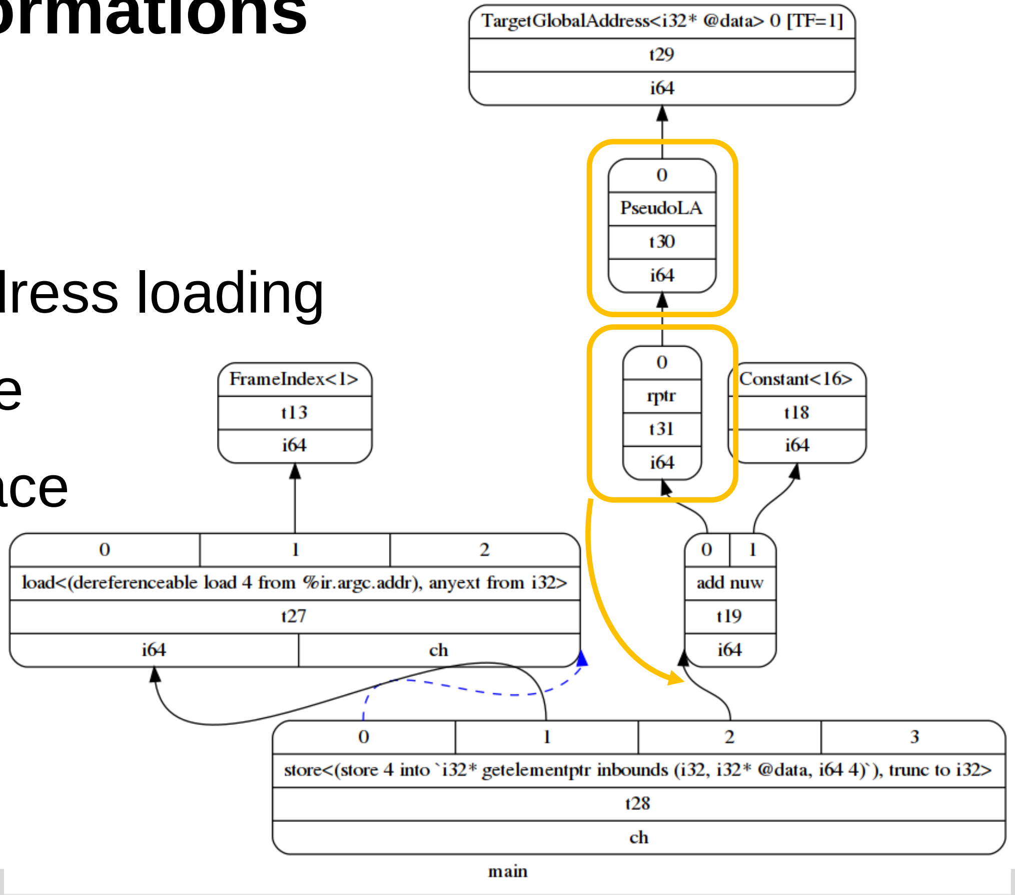
Robert Schilling[1,2], Mario Werner[1], Pascal Nasahl[1], Stefan Mangard[1]

[1]Graz University of Technology, [2]Know-Center GmbH

December 06th, 2018

Selection DAG Transformations

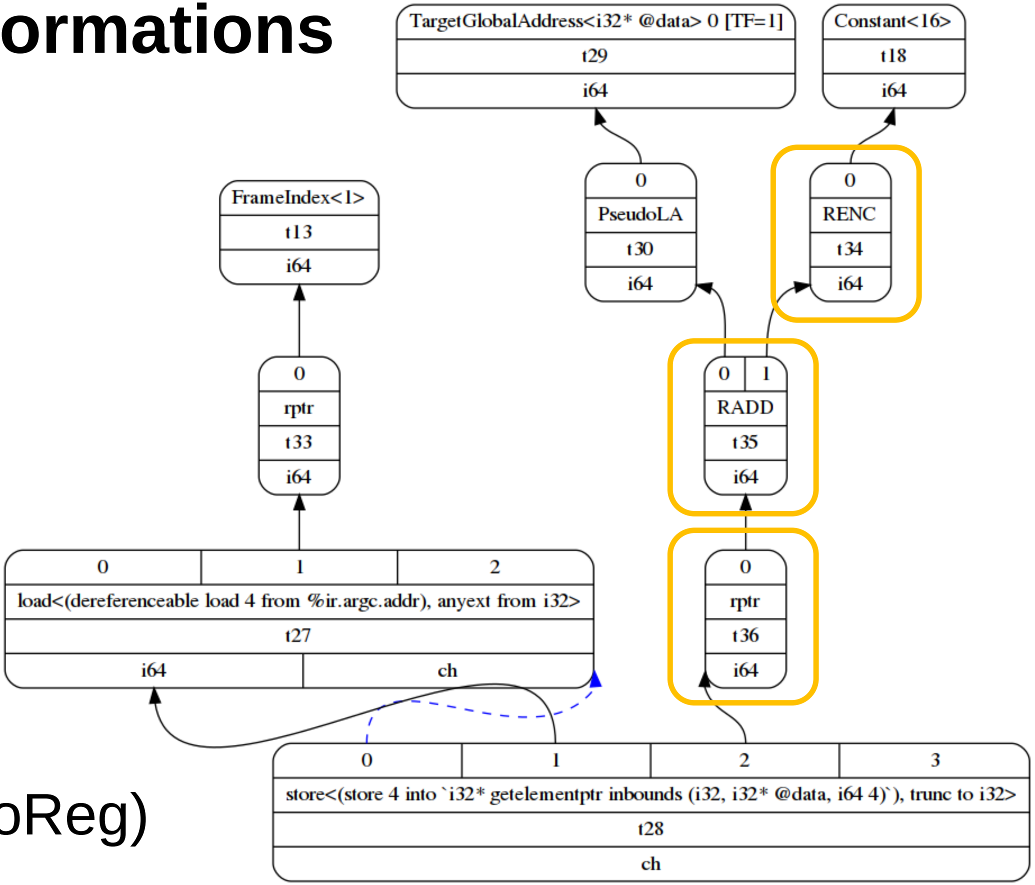
- Add PseudoLA
  - Used for custom address loading
- *rptr* node to track residue
- Propagate *rptr* and replace instruction


TargetGlobalAddress<i32* @data> 0 [TF=1]
t29
i64
0
PseudoLA
t30
i64
FrameIndex<1>
t13
i64
0
rptr
t31
i64
Constant<16>
t18
i64
0 1 2
load<(dereferenceable load 4 from %ir.argc.addr), anyext from i32>
t27
i64
ch
0 1
add nuw
t19
i64
0 1 2 3
store<(store 4 into `i32* getelementptr inbounds (i32, i32* @data, i64 4)`), trunc to i32>
t28
ch
main

Selection DAG Transformations

- *rptr* propagated over add
- Replace add with *RADD*
- Encode parameters

- Propagate from sources (PseudoLA, CopyFromReg) to sinks (loads/stores/CopyToReg)


TargetGlobalAddress<i32* @data> 0 [TF=1]
t29
i64
Constant<16>
t18
i64
FrameIndex<1>
t13
i64
0
PseudoLA
t30
i64
0
RENC
t34
i64
0
rptr
t33
i64
0 1
RADD
t35
i64
0 1 2
load<(dereferenceable load 4 from %ir.argc.addr), anyext from i32>
t27
i64 ch
0
rptr
t36
i64
0 1 2 3
store<(store 4 into `i32* getelementptr inbounds (i32, i32* @data, i64 4)`), trunc to i32>
t28
ch