A case against indirect jumps for secure programs

Alexandre Gonzalvez [1,2] **Ronan Lashermes** [3]

[1]IMT Atlantique [2]INRIA/TAMIS [3]INRIA/SED&LHS

November 21th, 2019

SILM Workshop, Rennes

Instructions

```
addi   a3, a0, 4
addi   a4, x0, 1
bltu    a4, a1, 10
jalr    x0, x1, 0
lw      a6, 0(a3)
addi   a2, a3, 0
addi   a5, a4, 0
lw      a7, -4(a2)
```

Atoms of our programs

Work on registers and offsets (encoded in the instruction value).

- Arithmetic
- Load/Store
- (Conditional) Branches
- (Inconditional) Jumps
- ...

The instructions are often seen as the output of the compiler.

Instructions

```
addi   a3, a0, 4
addi   a4, x0, 1
bltu    a4, a1, 10
jalr    x0, x1, 0
lw      a6, 0(a3)
addi   a2, a3, 0
addi   a5, a4, 0
lw      a7, -4(a2)
```

Atoms of our programs

Work on registers and offsets (encoded in the instruction value).

- Arithmetic
- Load/Store
- (Conditional) Branches
- (Inconditional) Jumps
- ...

The instructions are often seen as the output of the compiler.

In this talk, we will modify instructions to ease control flow graph extraction.

Context

Instruction Set Architecture

The definition of the instructions and possible configurations of our "machine" is the **instruction set architecture (ISA)**.

Security

The ISA definitions are mostly not concerned with security, apart for some forms of memory isolation (TrustZone, SGX).

Context

Instruction Set Architecture

The definition of the instructions and possible configurations of our "machine" is the **instruction set architecture (ISA)**.

Security

The ISA definitions are mostly not concerned with security, apart for some forms of memory isolation (TrustZone, SGX).

Avoiding legacy

Existing ISA have to ensure retro-compatibilty, which is bad for security. We are academics, we don't have to deal with that: we live in a fairy tale !

Our team's goal: designing ISAs for security

Functional vs formal ISA

Random number generators:

- No instructions present for a long time $\rightarrow$ not in the ISA scope.
- When present, not trusted (with good reasons).
- How do you verify TRNG output ? $\rightarrow$ **you can't**.

Our team's goal: designing ISAs for security

Functional vs formal ISA

Random number generators:

- No instructions present for a long time $\rightarrow$ not in the ISA scope.
- When present, not trusted (with good reasons).
- How do you verify TRNG output ? $\rightarrow$ **you can't**.

A new contract

The ISA must be seen as a contract between software and hardware. We are trying to add security properties in this contract.

Our team's goal: designing ISAs for security

Functional vs formal ISA

Random number generators:

- No instructions present for a long time → not in the ISA scope.
- When present, not trusted (with good reasons).
- How do you verify TRNG output ? → **you can't**.

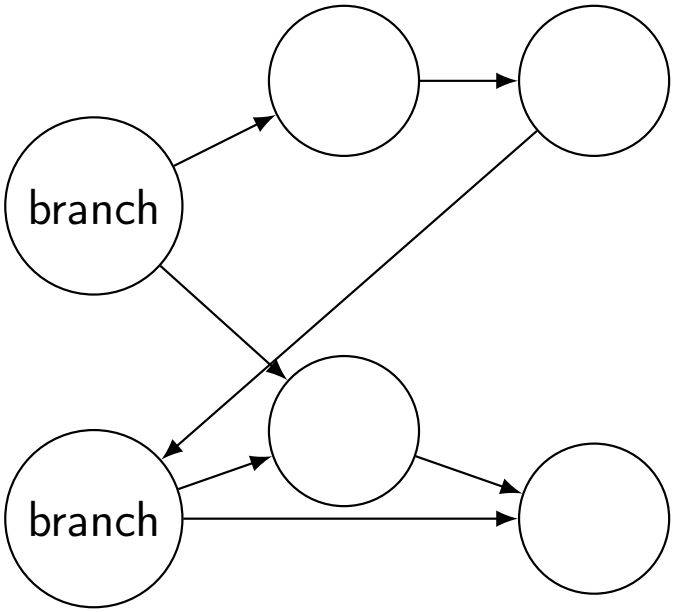
A new contract

The ISA must be seen as a contract between software and hardware. We are trying to add security properties in this contract.

Enabling CFI

In this talk, we will focus on the requirements to guarantee **control flow integrity (CFI)**.

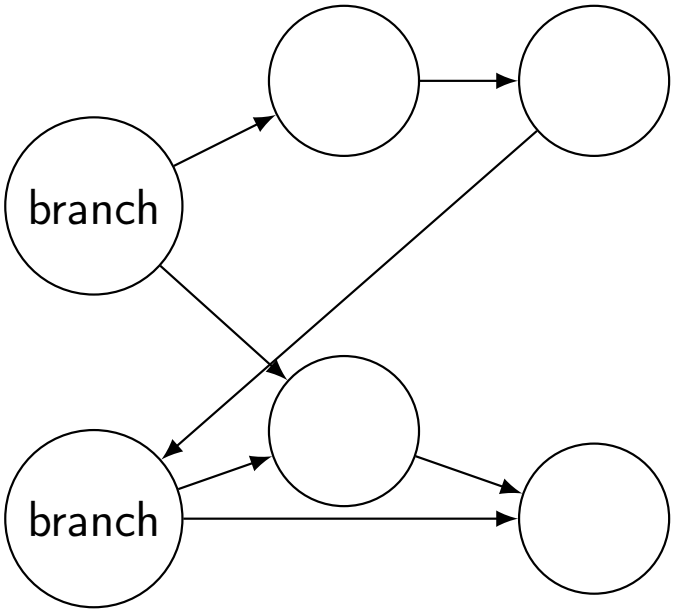
Control Flow Graph



Graph

The CFG is a graph, where nodes are instructions (including addresses) and edges are legal transitions between instructions.

Control Flow Graph



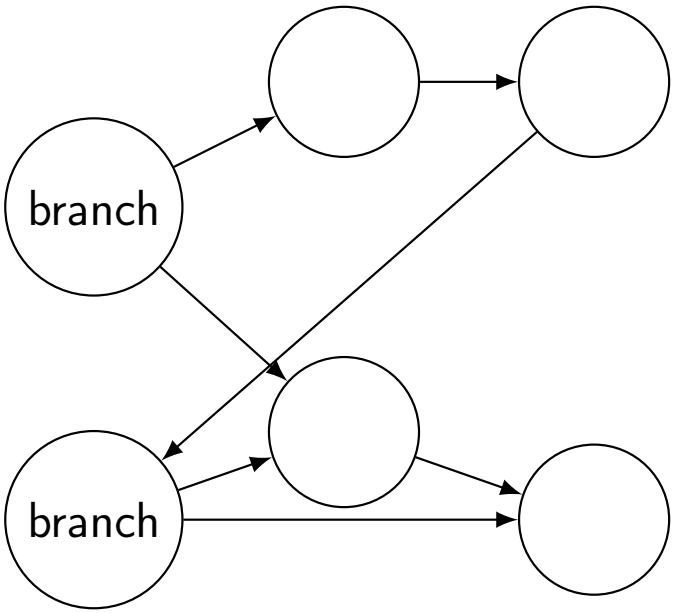
Graph

The CFG is a graph, where nodes are instructions (including addresses) and edges are legal transitions between instructions.

Indirect jumps

`jalr x0, x1`: jump to address present in `x1`. CFG extraction requires dataflow analysis.

Control Flow Graph



Graph

The CFG is a graph, where nodes are instructions (including addresses) and edges are legal transitions between instructions.

Indirect jumps

`jalr x0, x1`: jump to address present in `x1`. CFG extraction requires dataflow analysis.

CFG quality measure: precision ≈ coverage

- A CFG is always defined.
- Coverage is the ratio of nodes/edges followed during one execution.

Control Flow Integrity (CFI)

Informal definition

CFI guarantees that an attacker cannot alter the control flow.

Formal definition

CFI guarantees that an instruction can only be followed by another one that is targeted by an edge in the CFG.

Some mechanisms for CFI

- Shadow stack: **duplicate** the return addresses in another stack.
- Do some verifications for all indirect jumps (lots of variants).
- Instruction Set Randomization (ISR): encrypt the program with CFG information included.

Either self validation, or validation with respect to CFG.

Content

The problem: CFG extraction

A precise CFG cannot be extracted in general.

Trapdoor predicate

Let k be a secret key and h a cryptographic hash function.
Then we define the trapdoor predicate p as:

$$p(x) = \begin{cases} 1 & \text{if } h(x) = h(k) \\ 0 & \text{in the other cases} \end{cases}$$

Hiding the control flow

$$\begin{aligned} x &\leftarrow \text{user input} \\ y &\leftarrow p(x) \cdot (h(x+1) \oplus \mathit{constant}) \\ \mathit{jump} &\quad\; 0x1000 \oplus y \end{aligned}$$

Hiding the control flow

Hiding snippet

$$\begin{aligned} x &\leftarrow \text{user input} \\ y &\leftarrow p(x) \cdot (h(x+1) \oplus constant) \\ jump & \quad\; 0x1000 \oplus y \end{aligned}$$

Properties

Without the knowledge of k, it is impossible to:

- find x such that we jump to an address different than $0x1000$.
- find the destination address, if we jump to somewhere different than $0x1000$.

Removing indirect jumps

Indirect jumps considered evil ?

Instructions with potentially high fanout. Convert data uncertainty into control flow uncertainty.

Removing indirect jumps

Indirect jumps considered evil ?

Instructions with potentially high fanout. Convert data uncertainty into control flow uncertainty.

Why not just forbid them ?

Removing indirect jumps

Indirect jumps considered evil ?

Instructions with potentially high fanout. Convert data uncertainty into control flow uncertainty.

Why not just forbid them ?

Common answers

- Some programs become impossible to write ! (Virtual method tables for OO polymorphism. . .)
- Programs become inefficient !

Removing indirect jumps

Indirect jumps considered evil ?

Instructions with potentially high fanout. Convert data uncertainty into control flow uncertainty.

Why not just forbid them ?

Common answers

- Some program patterns become impossible to write $\rightarrow$ the unsecure ones !
- Dispatchers may be inefficient.

The machine

A same machine for 3 ISAs variations

A 64-bit RISC Harvard machine (D and I separated):

- Generict registers (RW): `x1`,$\cdots$, `x16`.
- Stack pointer (RW): `sp` for the data stack.
- Constant registers (R): `zero` and `full` ($x \oplus \mathit{full} = \neg x$).
- Program counter is a special register neither (directly) readable nor writable.
- 2 finite address spaces (for instructions and data), word addressed.

Setup

The program is written in a read-only instruction memory before machine boot.

Our ISAs

- ISAv1: base ISA with indirect jumps.
- ISAv2: no indirect jumps at all.
- ISAv3: backward indirect jumps allowed.

Our ISAs

- ISAv1: base ISA with indirect jumps.
- ISAv2: no indirect jumps at all.
- ISAv3: backward indirect jumps allowed.

Instructions

- Integer arithmetic (2 to 1): $+, -, \oplus, \wedge, \vee, \cdot, /,$ mod $, <<, >>$.
- Conditional branches: $=, \neq, \leq, <, \geq, >$. If condition valid, branch to offset.
- Direct load/store: load or store a data in memory to/from a register.
- Indirect load/store: load or store a data in memory at address in a register to/from another register.
- Load immediate: set the value of the given register (source of constants).
- Register move: copy a register into another.

Our ISAs

- ISAv1: base ISA with indirect jumps.
- ISAv2: no indirect jumps at all.
- ISAv3: backward indirect jumps allowed.

Instructions (continued)

- Non deterministic: load a non-deterministic value into a register. The semantic does not specify the source (TRNG, user input, etc).
- Halt: terminate the computation.
- Direct jump: go to address contained in the instruction.
- Call: direct jump to specified address and store the value of PC in a register (x14 for ISAv1) or a dedicated stack (ISAv3).
- Return: jump to the address popped from the dedicated return stack (ISAv3 only).
- Indirect jump: go to the address present in the given register (ISAv1 only).

Benchmark: AES

Need for security. Mix of various instruction types. Indirect jumps in our sbox implementation and to call procedures.

	ISAv1		ISAv2		ISAv3	
Type	Count	Ratio	Count	Ratio	Count	Ratio
Arithmetic	10011	0.44	9978	0.41	9825	0.44
LoadImmediate	5232	0.23	6767	0.27	5218	0.23
IndirectLoad	2650	0.12	2628	0.11	2554	0.11
IndirectStore	2464	0.11	2441	0.10	2368	0.11
DirectJump	668	0.03	719	0.03	668	0.03
IndirectJump	616	0.03	X	X	616	0.03
ConditionalBranch	601	0.03	1703	0.07	601	0.03
RegisterMove	370	0.02	370	0.02	370	0.02
NonDeterministic	2	0.00	2	0.00	2	0.00
Halt	1	0.00	1	0.00	1	0.00
Total	22615	1.00	24609	1.00	22223	1.00

Benchmark: AES

CFG extraction algorithms

- Structural (s): just follow the semantic content of the instruction opcodes (branches have two successors, etc.).
- Tracking (t): track the return stack in addition to the structural method to find `Return` destinations (easy without forward indirect jumps).

	Mean dur	Std dev	CFG nodes	nodes cov	CFG edges	edges cov
ISAv1 s	22635	24.1	778	99.6%	12437	6.4%
ISAv2 s	24605	24.1	816	99.1%	836	99.0%
ISAv3 s	22231	23.8	747	99.7%	11942	6.4%
ISAv3 t	22231	24.0	745	100%	764	100%

Dispatchers

The problem

Branch to *n* different destinations depending on a data value in a register.

Listing 1: ISAv2 dispatcher pattern

```
//x1 contains the value that decides where to branch (x1 in
    [0, 3])
load   x15, #0
beq    proc0, x1, x15
load   x15, #1
beq    proc1, x1, x15
load   x15, #2
beq    proc2, x1, x15
load   x15, #3
beq    proc3, x1, x15
jump   error_handler
```

Dispatchers costs

Instructions count

	Branch	Call	Return	Total
ISAv1	0	1	$3 \cdot p$	$3 \cdot p + 1$
ISAv2	$2 \cdot p$	0	p	$3 \cdot p$
ISAv3	$2 \cdot p$	$2 \cdot p$	p	$5 \cdot p$

Latency

	Branch	Call	Return	Total
ISAv1	0	1	3	4
ISAv2	$2 \cdot \lceil \log_2(p) \rceil + 1$	0	1	$2 + 2 \cdot \lceil \log_2(p) \rceil$
ISAv3	$2 \cdot \lceil \log_2(p) \rceil + 1$	2	1	$4 + 2 \cdot \lceil \log_2(p) \rceil$

Dispatchers, a conclusion

Branch based dispatchers are not efficient

The dispatcher latency is high and depends on the number of destinations $\rightarrow$ timing leakage.

Dispatchers, a conclusion

Branch based dispatchers are not efficient

The dispatcher latency is high and depends on the number of destinations $\rightarrow$ timing leakage.

A possible solution: n-fanout conditional branches

```
nbranch x1, #4
jump    proc0
jump    proc1
jump    proc2
jump    proc3
jump    error_handler
```

Accepted programs

All self-contained programs can be written with the new idioms.

Program structure reflects CFG

With ISAv2 and ISAv3, the instruction opcodes directly encodes the CFG (branchs have 2 successors, ...). An unprecise CFG means a bigger, inefficient, program.

- No dataflow analysis required to extract CFG from an ISAv2 or ISAv3 program.
- The program reflects its CFG, information can be used for CFI, in particular in hardware.

Rejected patterns

Some examples

We cannot write a program that can jump outside of itself.

- The kernel cannot instantiate a new user processus.
- An application cannot use plugins for additional functionalities.
- Dynamic code generation impossible.

Rejected patterns

Some examples

We cannot write a program that can jump outside of itself.

- The kernel cannot instantiate a new user processus.
- An application cannot use plugins for additional functionalities.
- Dynamic code generation impossible.

Is it a bad thing ?

Rejected patterns

Some examples

We cannot write a program that can jump outside of itself.

- The kernel cannot instantiate a new user processus.
- An application cannot use plugins for additional functionalities.
- Dynamic code generation impossible.

Is it a bad thing ?

An indirect jumps implies to switch to a new security domain

Indirect jumps should not be just instructions: a mechanism is required to switch to a new security domain.

Wrap-up

1. Indirect jumps are not required in most cases.
2. The efficiency cost without them is low even for dispatchers if we add a new high fanout branch instruction.
3. Without them, the program structure reflects the CFG $\rightarrow$ allows efficient CFI schemes.
4. An indirect jumps implies to switch to a new security domain.

Thank you!

Any questions?