

"Who ya gonna call?" Cybersecurity for the Spectre Era

Calvin Deutschbein

...the Spectre Era

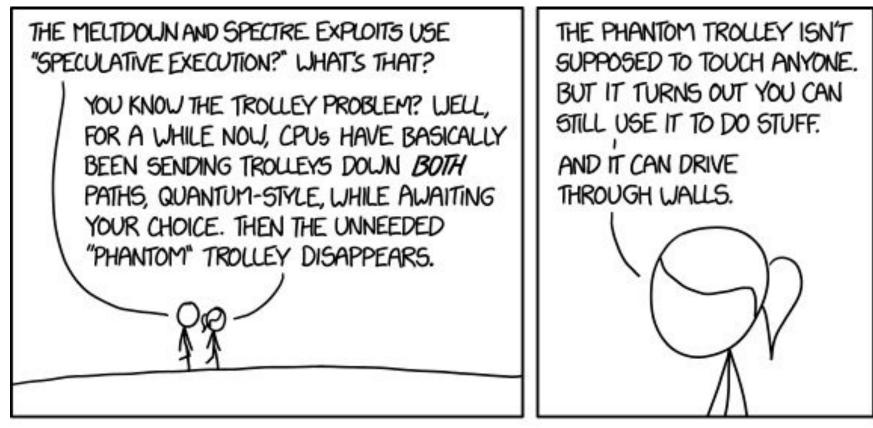
3 Jan, 2018: Google Project Zero et al. publicly report the Spectre vulnerability.

- Spectre targets **hardware** (all Intel processors since 1995)
- Spectre leaves **no traces** in traditional logs
- Spectre went undetected for over two decades

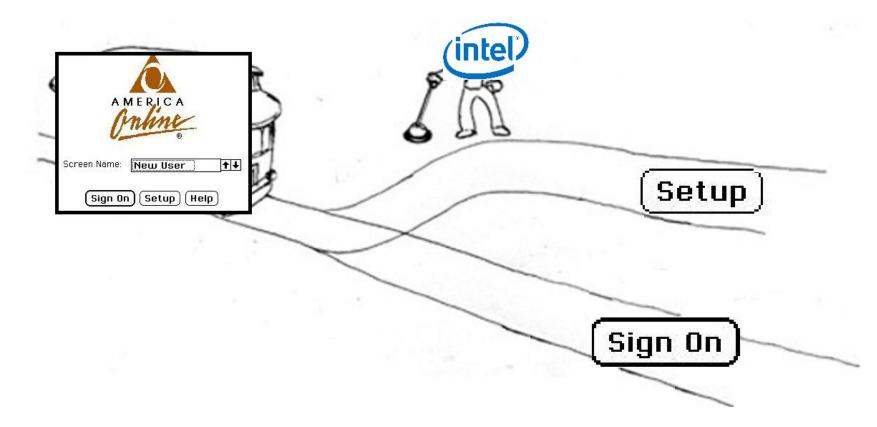
Learn more: meltdownattack.com/



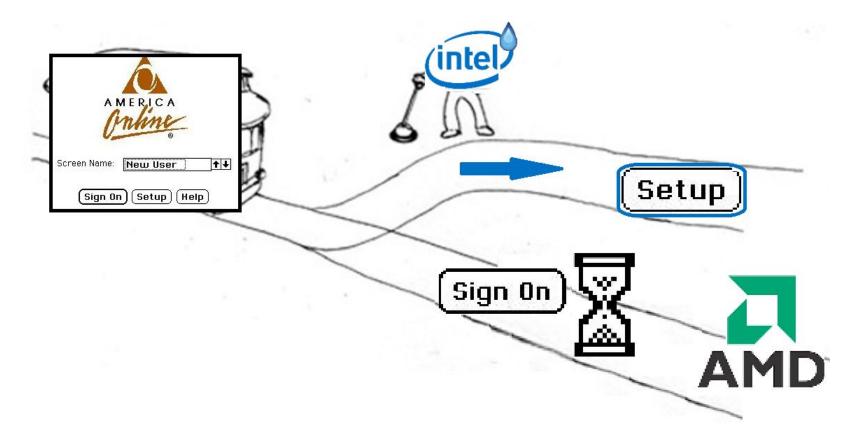
What is Spectre?



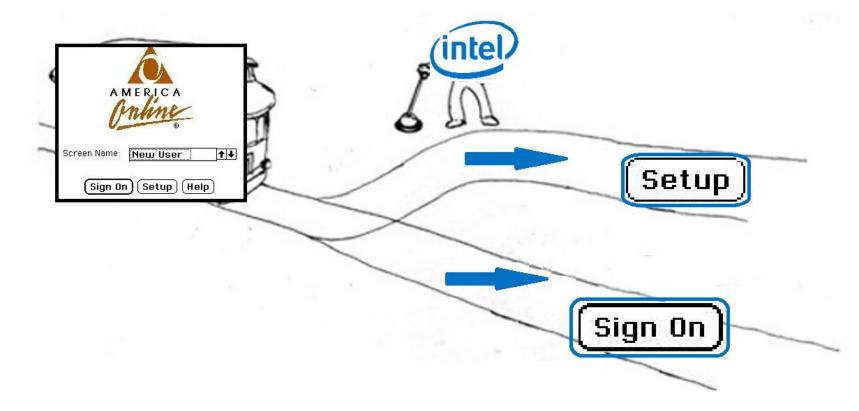
The Branch Prediction Problem in 1995



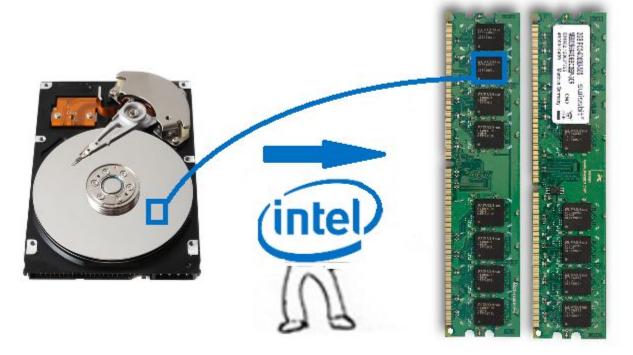
To avoid delays, x86 chips "guess" what's next



But what if chips guess an unused branch?

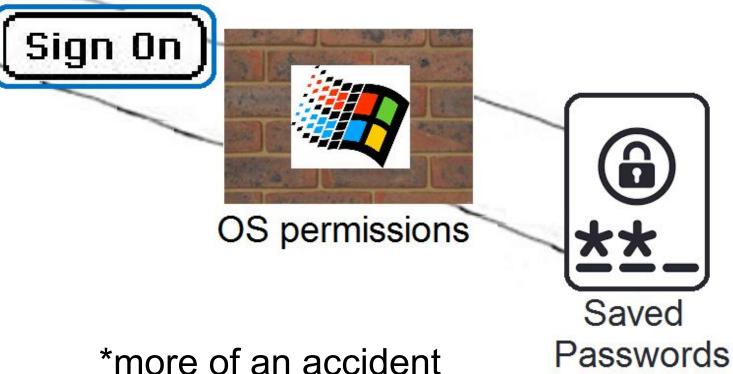


Speculative execution can access memory...*



*This is the point of speculative execution.

...and bypass memory protections*



Exploiting the Vulnerability

Spectre is a **vulnerability** - it provides an entry point for an adversary.

• Spectre: "Branch misprediction may leave observable side effects"

Adversaries must **exploit** the vulnerability to gain access to secure data.

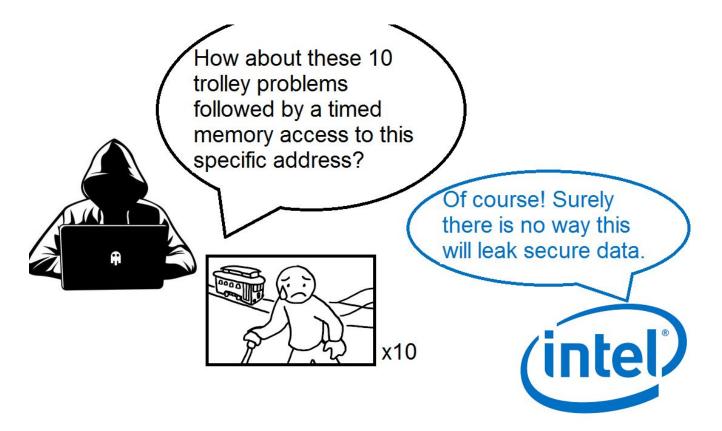
• Exploit: "Using observable side effects to access secure data."

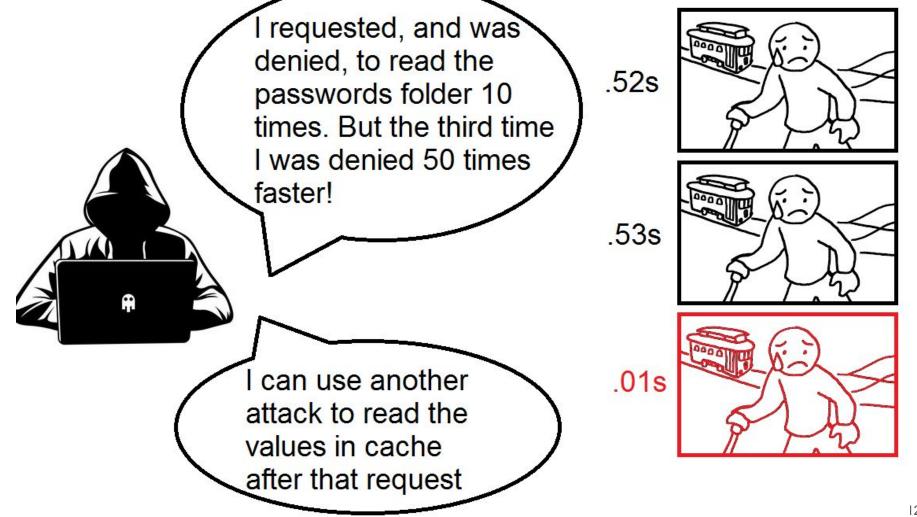
The mere presence of the vulnerability in on hardware not running code capable of exploiting the vulnerability will not result in a security violation.

How adversaries can exploit Spectre

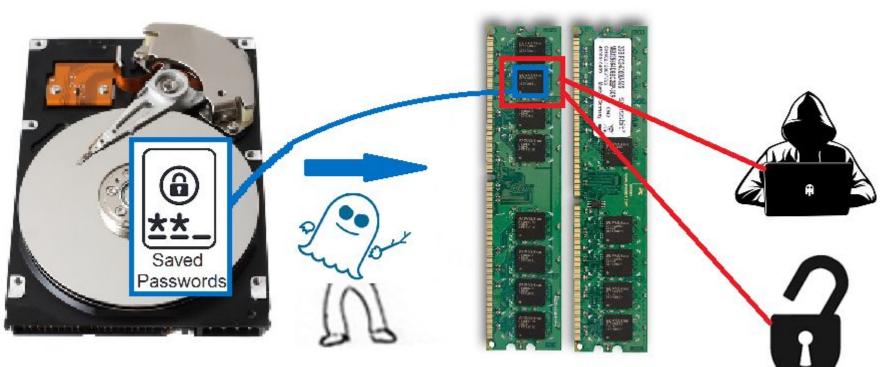


How adversaries can exploit Spectre





Thus, an attacker gains access to secure data



Three attack surfaces:

• Timing

• Timing cache hits

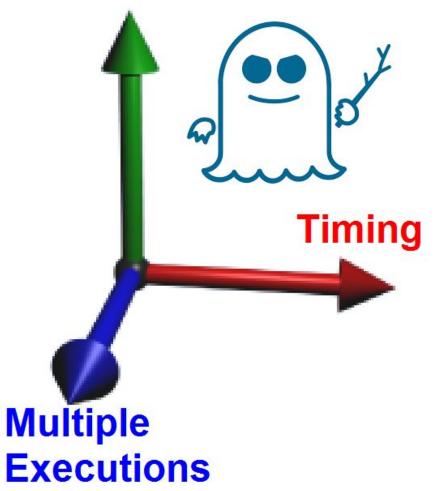
• Microarchitecture

- Branch prediction
- Below operating system or assembly

• Multiple Executions

• Multiple runs expose timing differences

Microarchitecture





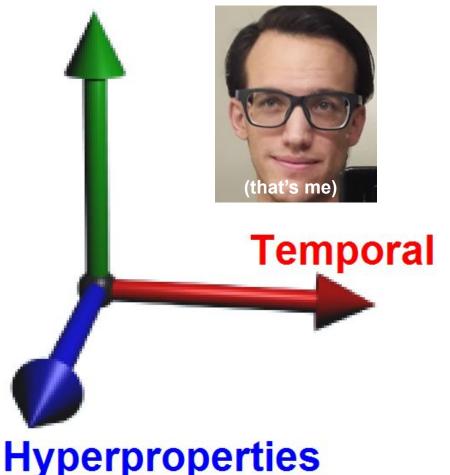
So... "Who ya gonna call?"

Mining Behavior

My research shows the technique of **specification mining** can find:

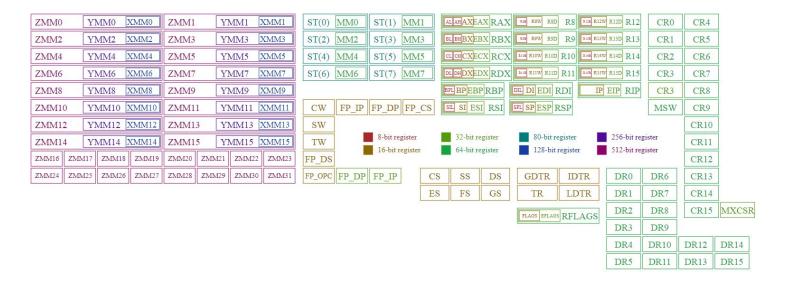
- Temporal properties, for timing
- **Closed source CISC** architecture properties, for microarchitecture
- **Hyperproperties**, properties over multiple traces of execution

Closed Source CISC



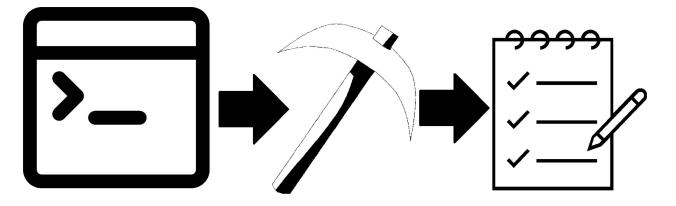
Defining Secure Behavior

For x86-64, no specification exists - so we create one.



Specification Mining

- Miners accept as input traces of execution.
 - For example, the debug output of an x86-64 processor booting Linux.
- Miners find **properties** that hold over the traces.
 - For example, "if reset is active, then the privilege level is supervisor".
 - RESET==0 \Rightarrow CURRENT_PRIVILEGE_LEVEL==0
- Miners contain powerful inference engines for high performance.

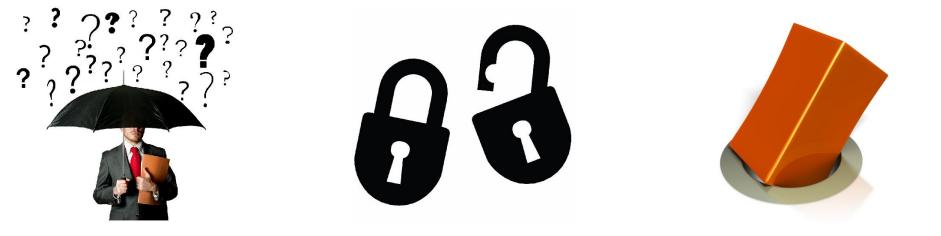


Undine: Mining Temporal Properties

Can **linear temporal logic** properties that model secure behavior be discovered using specification mining?

A library of typed templates for my miner, Undine, enable it to find security temporal properties, including properties using **G** (Globally) or **X** (Next) operators.

Difficulties Finding Security Properties



Too Many Properties Properties Not Security Related Do Not Capture Semantic Info

Without separate events there are many propertiesSample TraceMined 30 $G(x \rightarrow y)$ reg_a==1reg_a==1 \rightarrow reg_b==1reg_b==1...reg_c==0reg_a==1 \rightarrow reg_c==reg_d

...

reg d==0

reg_a==reg b

reg c==reg d

 $reg_c = reg_d \rightarrow reg_a = reg_b$

Templates Refine to Security Properties Sample Trace Mined 8 G($R \rightarrow R-R$) reg a==1 \rightarrow reg a==reg b reg a == 1reg a==1 \rightarrow reg c==reg d reg b==1 reg c==0 reg b==1 \rightarrow reg a==reg b reg d==0 ... reg f==0 \rightarrow reg c==reg d reg a==reg b reg c==reg d

Register Roles Refine Further

Sample Trace

reg a==1

reg b==1

Mined 2 G($\mathbb{R} \rightarrow \mathbb{R}\text{-}\mathbb{R}$)

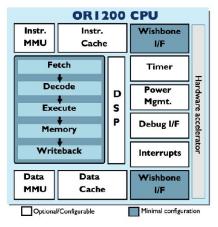
 $reg_a==1 \rightarrow reg_c==reg_d$ $reg_b==1 \rightarrow reg_c==reg_d$

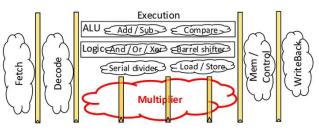
reg_c==reg_d

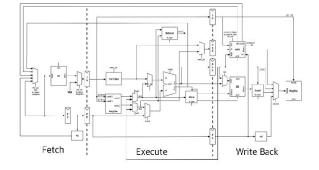
Register Slices Uncover Semantic Meaning Sample Trace Mining G(a) <no properties> reg a = = 7#tick reg a==3 #tick reg a = = 5. . .

Register Slices Uncover Semantic Meaning Sample Trace Mining G(a) reg a[0]==1 reg a[0]==1 reg a[1]==1 #tick reg a[0]==1 reg a[1]==1 #tick reg a[0]==1 reg a[1]==0

Tested on 3 Processors







OR1200

mor1kx

RISC-V

Undine: Mining Temporal Properties

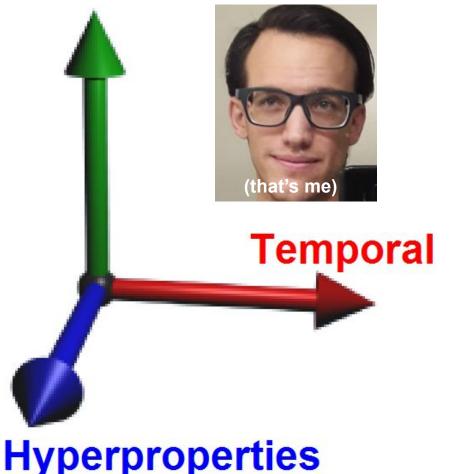
Undine can discover linear temporal logic security properties such as those related to correct initialization of a system using a library of typed templates.

Mining Behavior

My research shows the technique of **specification mining** can find:

- **Temporal** properties, for timing
- **Closed source CISC** architecture properties, for microarchitecture
- **Hyperproperties**, properties over multiple traces of execution

Closed Source CISC

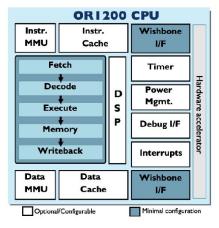


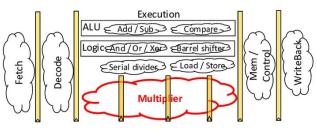
Astarte: Mining Closed Source CISC

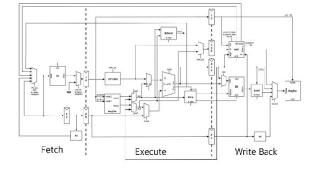
How can properties that model secure behavior of **closed source complex instruction set computer (CISC)** designs be discovered using specification mining?

Mining for control signals in the design then mining preconditioned on those control signals yields security properties of the design.

Recall: Undine Tested on 3 Processors







OR1200

mor1kx

RISC-V

All were Open Source and RISC! x86 is neither!

ZMM0	Y	MM0 [XMM0	ZMM1	YI	MM1	XMM1	ST(0)	MM0	ST(1)	MM1		AL AHAXEA	AXRAX	RIB RSW RSD	R8 R12B R1	2W R12D R12	CR0	CR4	
ZMM2	Y	MM2 [XMM2	ZMM3	Y	MM3	XMM3	ST(2)	MM2	ST(3)	MM3		BL BHBXE	BX RBX	ROB ROW ROD	R9 R138 R1	3W R13D R13	CR1	CR5]
ZMM4	Y	MM4 [XMM4	ZMM5	Y	MM5	XMM5	ST(4)	MM4	ST(5)	MM5		CL CHCXEO	CX RCX	RION RION	R10 R148 R14	4W R14D R14	CR2	CR6]
ZMM6	Y	MM6 [XMM6	ZMM7	YI	MM7	XMM7	ST(6)	MM6	ST(7)	MM7		dl dh DXEI	DX RDX	RIIB RIIW RIID	R11 RISE RI	SW RISD R15	CR3	CR7]
ZMM8	Y	MM8 [XMM8	ZMM9	Y	MM9	XMM9					E	BPL BP EBI	PRBP	DIL DI EDI	RDI I	P EIP RIP	CR3	CR8]
ZMM10	0 Y	MM10 [XMM10	ZMM1	1 YI	MM11	XMM11	CW	FP_IP	FP_DP	FP_C	S	SIL SI ES	I RSI	SPL SP ESP	RSP		MSW	CR9]
ZMM12	2 Y	MM12 [XMM12	ZMM1.	3 YI	MM13	XMM13	SW					-						CR10]
ZMM14 YMM14 XMM14			ZMM15 YMM15 XMM15			TW 8-bit register					32-bit re					gister	CR11]		
ZMM16	ZMM17	ZMM18	ZMM19	ZMM20	ZMM21	ZMM22	ZMM23	FP_DS					64-bit register		120-01 Tegister		512-bit re	gister	CR12]
ZMM24	ZMM25	ZMM26	ZMM27	ZMM28	ZMM29	ZMM30	ZMM31	FP_OPC	FP_DP	FP_IP		CS	SS	DS	GDTR	IDTR	DR0	DR6	CR13]
												ES	FS	GS	TR	LDTR	DR1	DR7	CR14	
															FLAGS EFLAG	RFLAGS	DR2	DR8	CR15	MXCSR

DR14

DR15

DR12

DR13

DR3

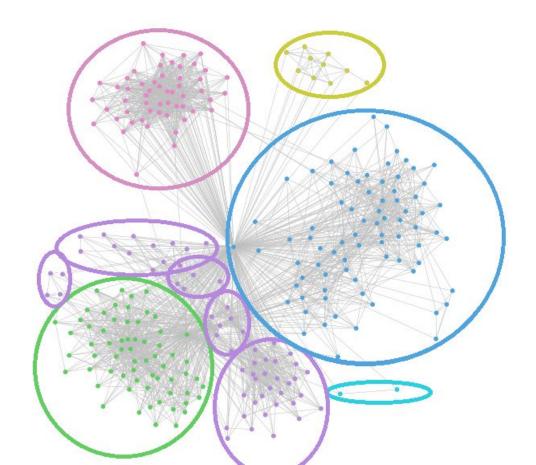
DR4 DR5 DR9 DR10

DR11

The x86 specification has many control signals...

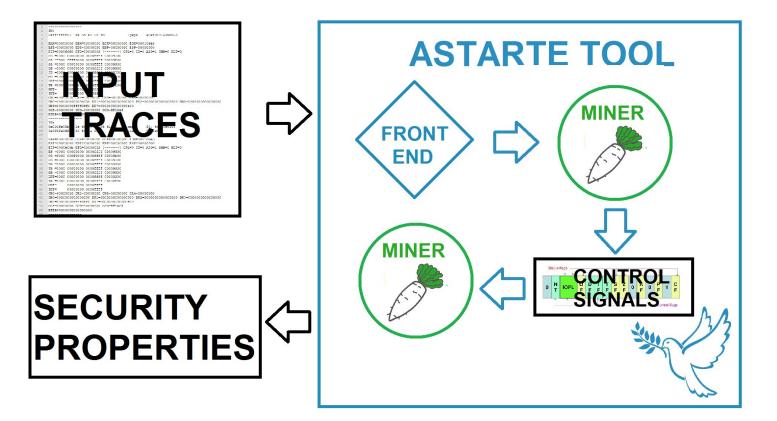
ZMM0	Y	MM0	XMM0	ZMM1	Y	MM1	XMM1	ST(0)	MM0	ST(1)	MM1		ALAHAXE	AXRAX	REB RSW RS	D R8 RIBR	112W R12D R12	CRO	CR4	
ZMM2	Y	MM2	XMM2	ZMM3	Y	MM3	XMM3	ST(2)	MM2	ST(3)	MM3		BL BHBXE	3X RBX	ROB ROW RO	D R9 2138 R	13W R13D R13	CR1	CR5	
ZMM4	Y	'MM4	XMM4	ZMM5	Y	MM5	XMM5	ST(4)	MM4	ST(5)	MM5		CLCHCXE	CX RCX	RIOB RIOW RIO	D R10 2148 R	14W R14D R14	CR2	CR6	
ZMM6	Y	MM6	XMM6	ZMM7	Y	MM7	XMM7	ST(6)	MM6	ST(7)	MM7		DLDHDXE	DX RDX	RIIB RIIW RII	D R11	15W RISD R15	CR3	CR7	
ZMM8	Y	MM8	XMM8	ZMM9	YI	MM9	XMM9						BPL BP EB	PRBP	DIL DI EDI	RDI	IP EIP RIP	CR3	CR8	
ZMM10	0 Y	MM10	XMM10	ZMM1	1 YI	MM11	XMM11	CW	FP_IP	FP_DP	FP_C	S	SIL SI ES	I RSI	SPL SP ESP	RSP		MSW	CR9	
ZMM12	2 Y	MM12	XMM12	ZMM1.	3 YI	MM13	XMM13	SW											CR10	
ZMM14	ZMM14 YMM14 XMM14		ZMM15 YMM15 XMM15				100000000000000000000000000000000000000	8-bit register 16-bit register		32-bit register 64-bit register		80-bit 128-bi	register	and the second of	256-bit register 512-bit register					
ZMM16	ZMM17	ZMM18	3 ZMM19	ZMM20 ZMM21 ZMM22 ZMM23					_	04-0111	egister	128-01	512-01116	J12-on register						
ZMM24	ZMM25	ZMM26	5 ZMM27	ZMM28	ZMM29	ZMM30	ZMM31	FP_OPC	FP_DP	FP_IP		CS	SS	DS	GDTR	IDTR	DR0	DR6	CR13	
												ES	FS	GS	TR	LDTR	DR1	DR7	CR14	
															FLAGS EFLA	S RFLAGS	DR2	DR8	CR15	MXCSR
																	DR3	DR9		
																	DR4	DR10	DR12	DR14
																	DR5	DR11	DR13	DR15

Control Signals Partition the Space

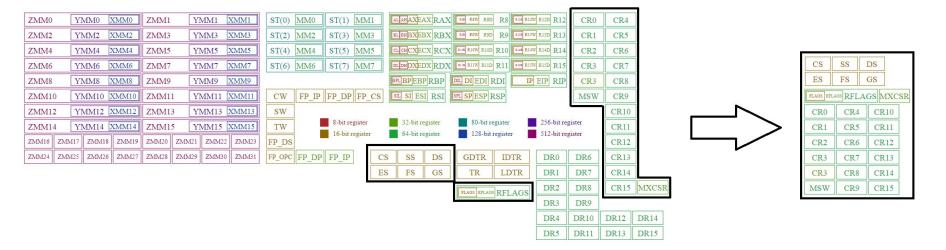


33

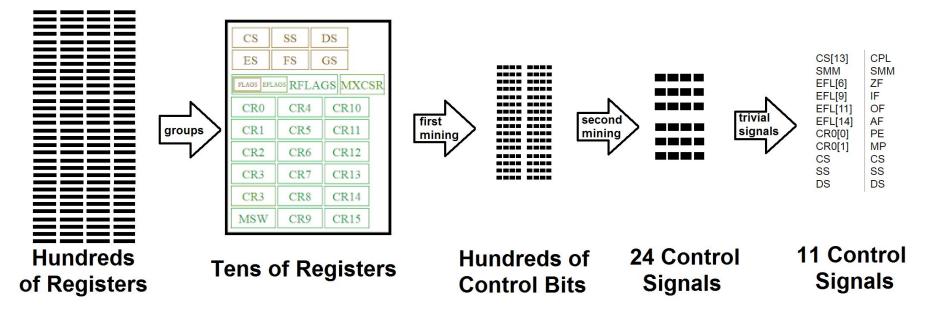
So I created a tool to find properties using signals.



Front End: Registers Placed in Groups



Property Refinement



Control Signals Partition the Space

Preconditions capturing changes to signals capture transitions between different modes of the processor. Preconditions holding signals constant capture the behavior defined by a control bit taking on a certain value.

Astarte: Mining Closed Source CISC

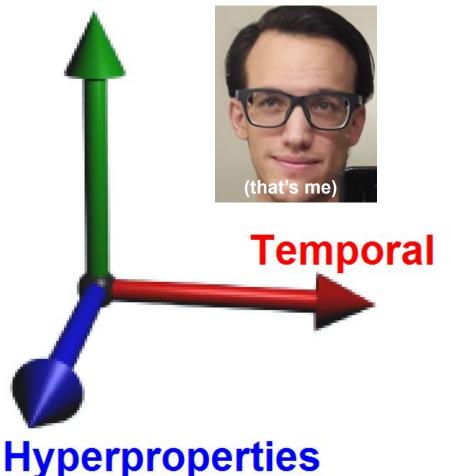
Specification mining can discover security properties preconditioned on control signals in closed source CISC designs.

Mining Behavior

My research shows the technique of **specification mining** can find:

- Temporal properties, for timing
- **Closed source CISC** architecture properties, for microarchitecture
- **Hyperproperties**, properties over multiple traces of execution

Closed Source CISC

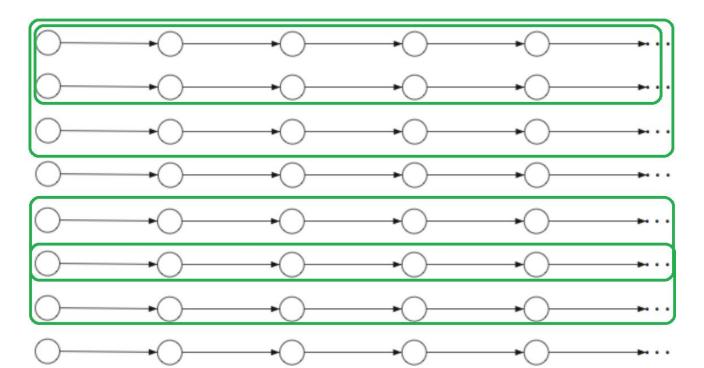


Isadora: Mining Hyperproperties (Current work)

How can **hyperproperties** that model secure behavior of designs be discovered using specification mining?

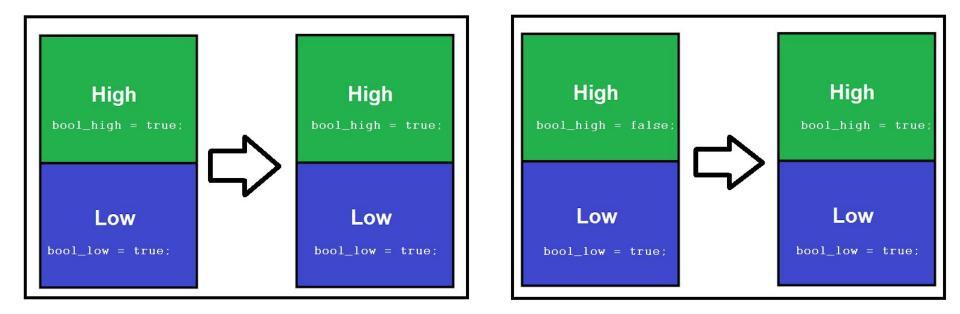
Hyperproperties

Sets of Sets of Traces, or Sets of Properties



41

Example: GMNI (Noninterference)

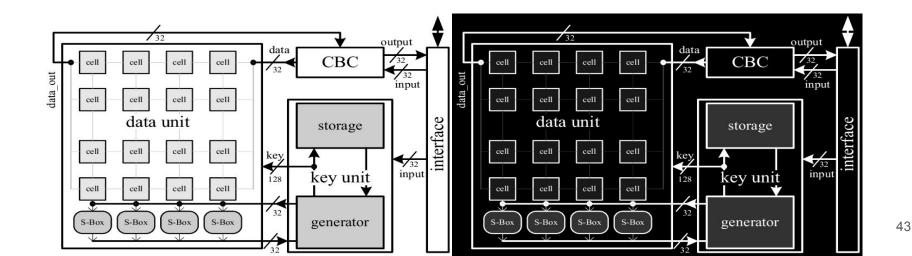


"High" could be OS, "Low" could be adversary

Instrumentation

To find hyperproperties, use Information Flow Tracking (IFT) instrumentation.

- IFT creates a shadow register for all design registers to track information flow
- GMNI is an information flow hyperproperty



Problem Statement

How can Information Flow Tracking (IFT) and specification mining determine

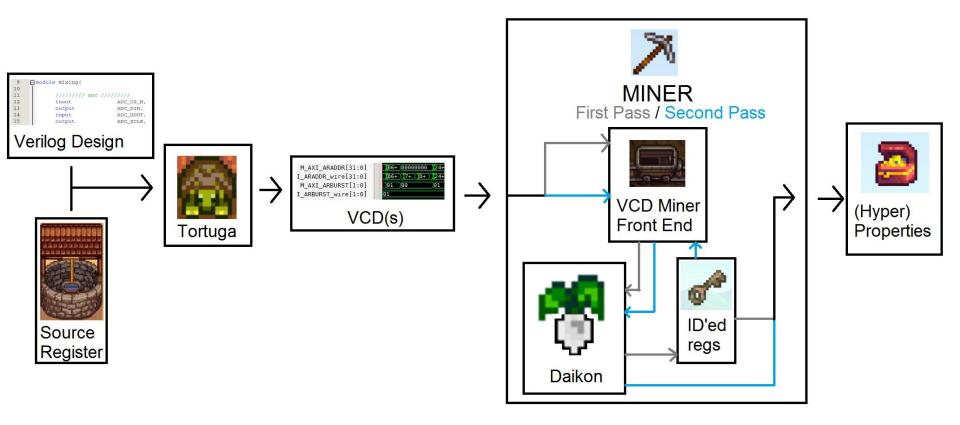
where and when

interference occurs in a design from any arbitrary source?

Tracking Information Flow

- Given a source, registers can be in one of three categories:
 - Always a sink: source = => sink ("flows to")
 - Never a sink: source =/=> sink ("does not flow to")
 - Conditionally a sink
 source =/=> sink UNLESS <boolean expression>

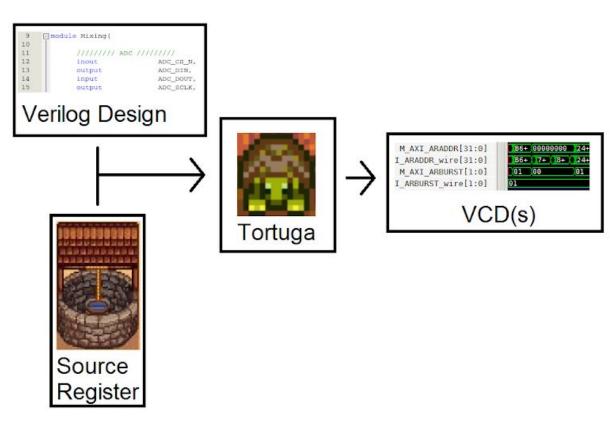
Research Technique Sketch



Trace Detail

More than traces!

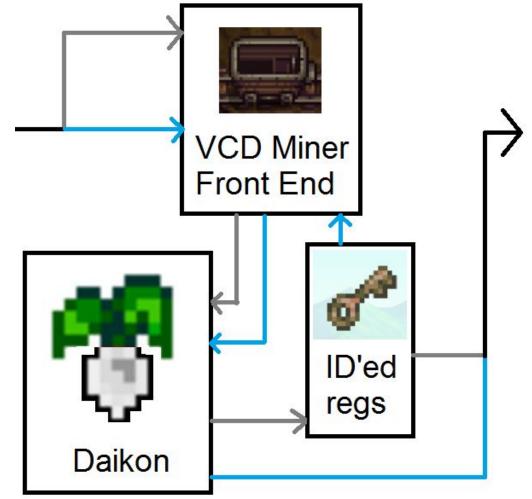
- 1. Specify Source
- 2. Generate Trace and IFT
- 3. Look at relevant regs



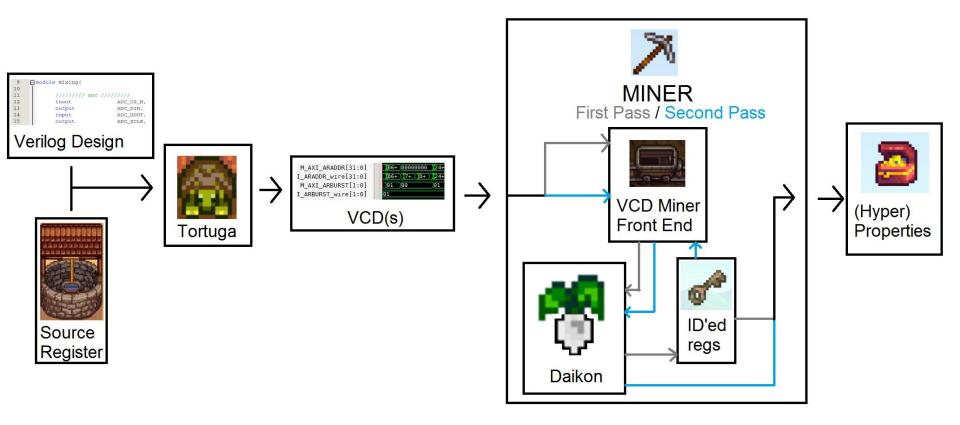
First Pass / Second Pass

Miner Detail

- 1. Input Traces
- 2. Run Miner
- 3. Get Output
- 4. Flag interesting shadow_*
 - a. shadow_* is IFT state
- 5. (Re-)Run Miner
- 6. Output Information Flow
 - a. "Always, never, maybe"



Research Technique Sketch



Mining in Practice

- Test using *write-address* register
 - Always sink 003 regs
 - Never sink 189 regs
 - Conditional sink 037 regs
- Secondary mining passes can determine conditions under which the 37 conditional sinks are affected by the source register

Isadora: Mining Hyperproperties (Current work)

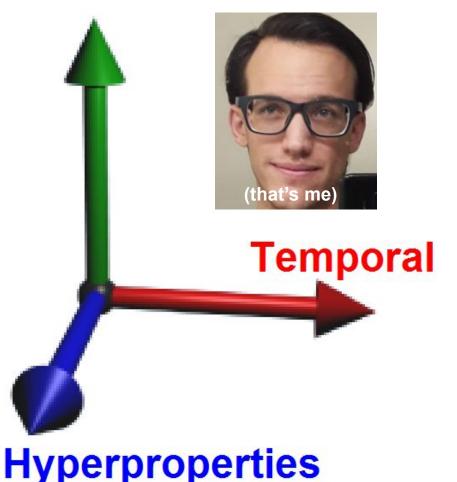
Hyperproperties that model secure behavior of designs be discovered using specification mining along with Information Flow Tracking (IFT).

Mining Behavior

My research shows the technique of **specification mining** can find:

- **Temporal** properties, such as correct initialization
- **Closed source CISC** architecture properties, such those over x86-64
- Hyperproperties, properties over multiple traces of execution

Closed Source CISC





"Who ya gonna call?" Cybersecurity for the Spectre Era

Any Questions?