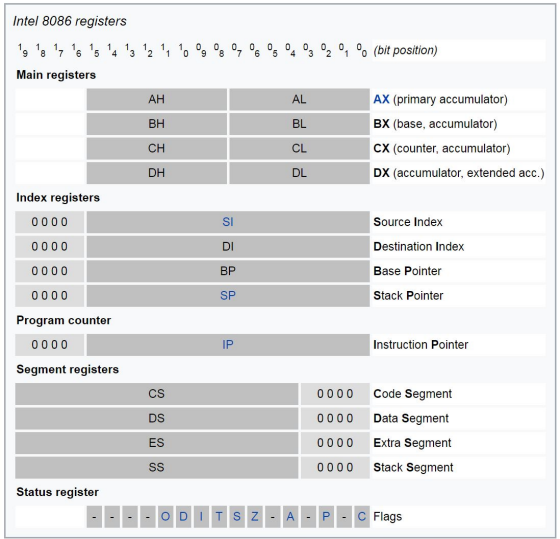
Thesis Proposal: Mining Secure Behavior of Hardware Designs

Calvin Deutschbein
Advised by Dr. Cynthia Sturton

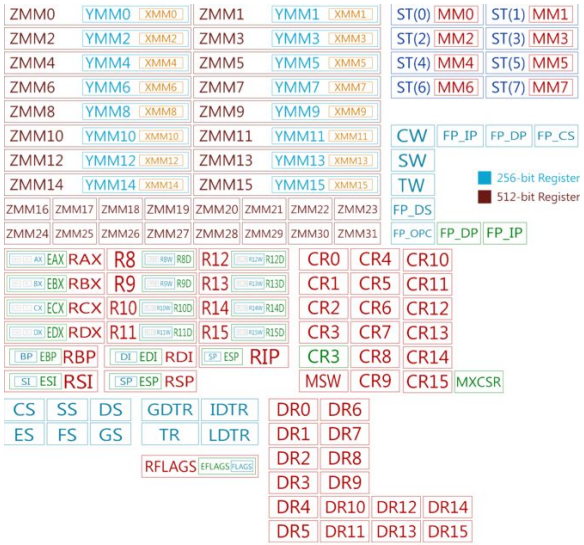
THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

Hardware is Growing Increasingly Complex

1979: 17 registers (8086)



2003: 209 registers (x86-64)



2019: *n*>209 registers (x86-64)

Model Specific Regs

Modules

Extensions

Graphics by Wikimedia

Attacks are Increasingly Difficult to Anticipate

Secure Behavior

Hardware attacks demand validating hardware designs:

- Symbolic Execution
- Deductive Verification
- Extensive Testing

All methods rely on working models of secure behavior.

Defining Secure Behavior

The state of the art for finding secure behavior is the SCIFinder tool (Zhang 2017).

- SCIFinder used processor errata as an initial set of security-critical invariants.
- SCIFinder used machine learning to infer an additional set of invariants.
- SCIFinder found 19/22 manually vulnerabilities, plus 3 new vulnerabilities.

SCIFinder finds propositional logic properties over open source RISC designs.

Mining Behavior

Models for secure behavior can be effective beyond propositional logic properties over open source RISC designs, such as:

- For **linear temporal logic** properties, such as correct initialization
- For **closed source CISC** designs, such as x86
- For **hyperproperties**, properties over multiple traces capturing side channels

Mining over designs can autonomously discover secure behavior.

Thesis

Specification mining can discover

- **linear temporal logic** security properties such as those related to correct initialization of a system,
- security properties preconditioned on control signals in **closed source CISC designs**, and
- security **hyperproperties** related to information flow.

Undine: Mining LTL Properties on RISC

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 known and new correct initialization properties.

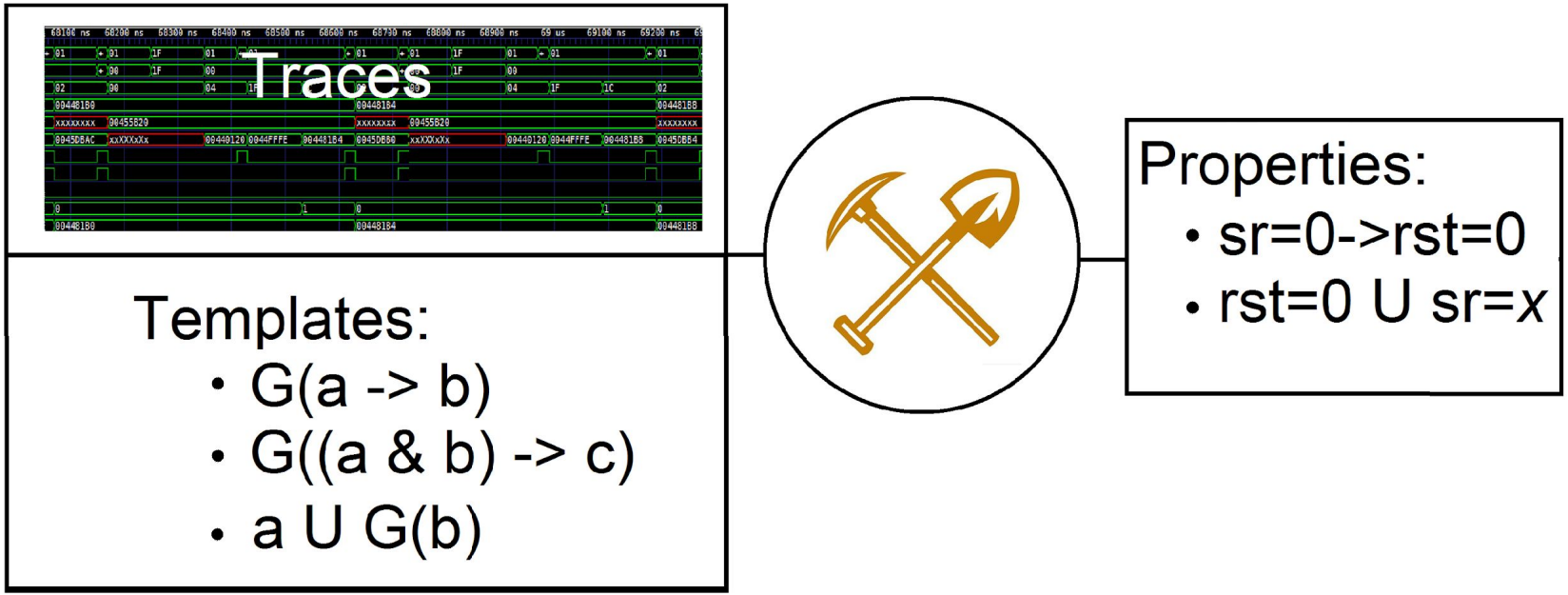
What are temporal properties?

Temporal properties allow the use of temporal operators, such as **G**lobally or **U**ntil.

For example, the may encode properties during initialization or across pipelines.

Known security properties from SCIFinder hold after reset, reset is not defined.

Specification Mining



Difficulties Finding Security Properties

Too Many Properties

Properties Not Security Related

Do Not Capture Semantic Info

How to define security temporal properties?

Undine creates a library of templates over:

- Known or expertise defined security signals
- In known or expertise defined temporal relations to each other

This library:

- Captures known properties from prior work
- Captures new properties over new architectures
- Refines property search to security properties.

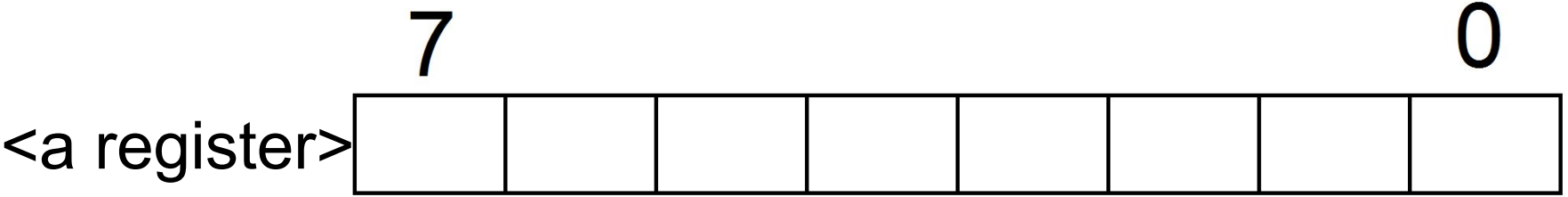
Solution in Action

Processor events, both directly from traces and derived, are categorized.

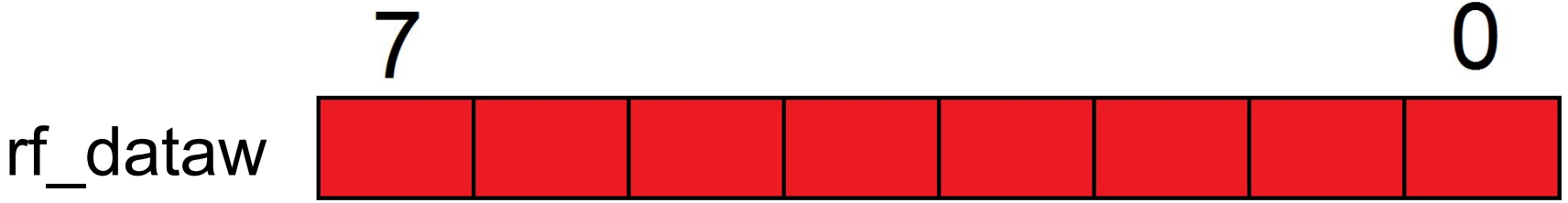
- Registers may be equal to some value.
- Some bits within a register may be equal to some value.
- Registers may satisfy some relation with each other.

These processor event categories are applied to the propositional variables in linear temporal logic templates.

Different processor events

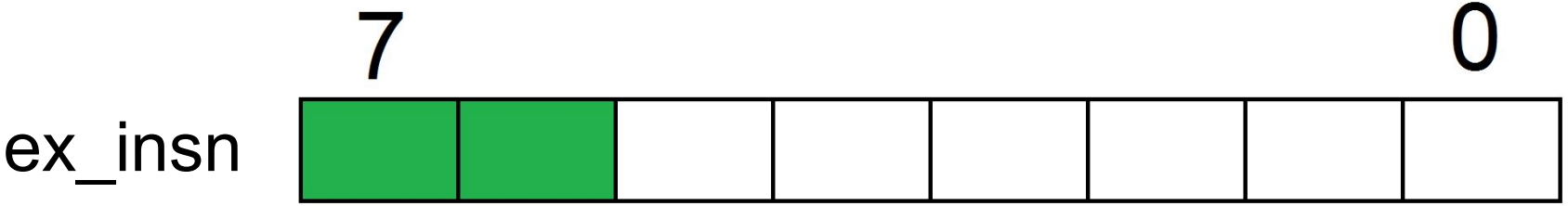


Register Value: Register simply holds some value



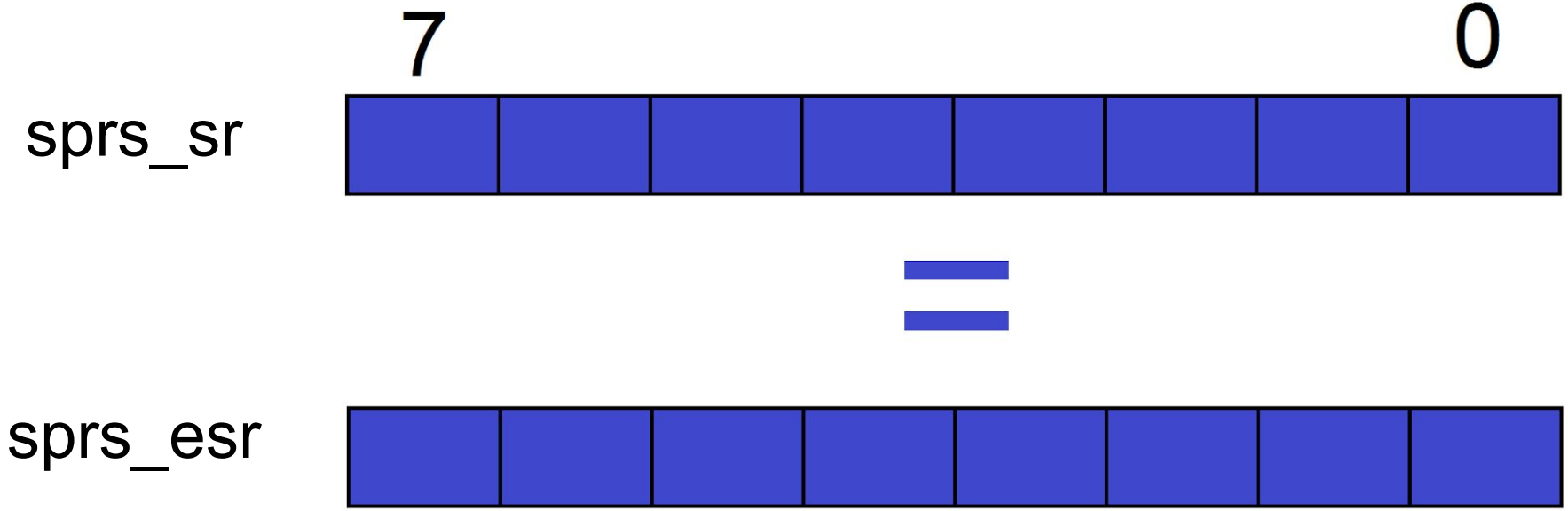
This gives the value to be written to a register.

Slice-Register: Semantic meaning of bits



This gives the opcode with no operands.

Register-Register: Two registers must be equal



This shows if status is saved during exceptions.

Known Properties + Expertise = Typed Templates

1	Register-Register U G(Register)
2	G(Slice-Register → Register-Register)
3	G((Slice-Register ∧ Slice-Register) → Register-Register)

Without typing there are many properties

Sample Trace

```
reg_a==1
reg_b==1
reg_c==0
reg_d==0
reg_a==reg_b
reg_c==reg_d
```

Mined 30 G(x → y)

```
reg_a==1 → reg_b==1
…
reg_a==1 → reg_c==reg_d
…
reg_c==reg_d → reg_a==reg_b
```

Typing events refines to security properties

Sample Trace

```
reg_a==1
reg_b==1
reg_c==0
reg_d==0
reg_a==reg_b
reg_c==reg_d
```

Mined 8 G(R → R-R)

```
reg_a==1 → reg_a==reg_b
reg_a==1 → reg_c==reg_d
reg_b==1 → reg_a==reg_b
…
reg_f==0 → reg_c==reg_d
```

Apply types to registers

Sample Trace

```
reg_a==1
reg_b==1
reg_c==reg_d
```

Mined 2 G(R → R-R)

```
reg_a==1 → reg_c==reg_d
reg_b==1 → reg_c==reg_d
```

Register Slices Uncover Semantic Meaning

Sample Trace

```
reg_a==7
#tick
reg_a==3
#tick
reg_a==5
...
```

Mining G(a)

<no properties>

Register Slices Uncover Semantic Meaning

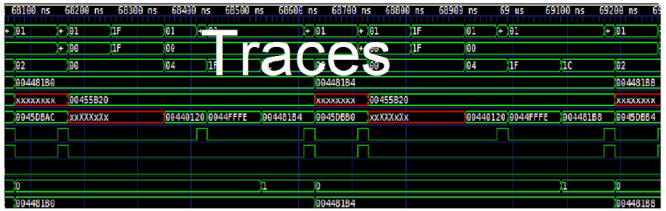
Sample Trace

```
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
```

Mining G(a)

```
reg_a[0]==1
```

UNDINE Implements Typed Template Mining



Typed LTL Templates:

- RR_a U G(R_b),
- G((SR_a ∧ SR_b) → R-R_c)
- G(SR_a → RR_b)

Type Information:

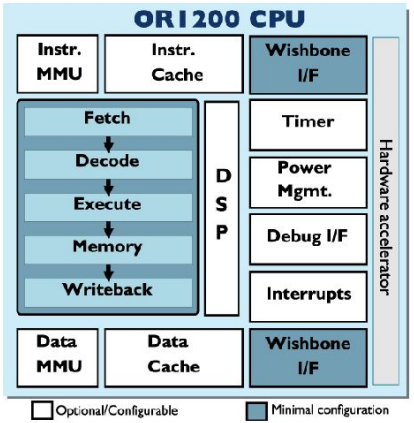
- type(rst) = R
- type(insn) = SR

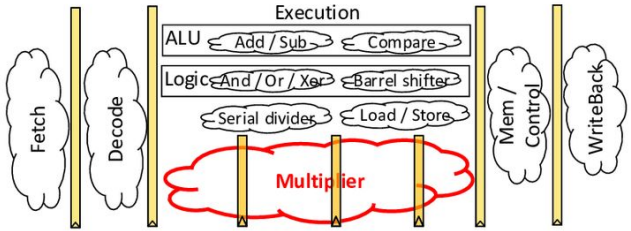
UNDINE

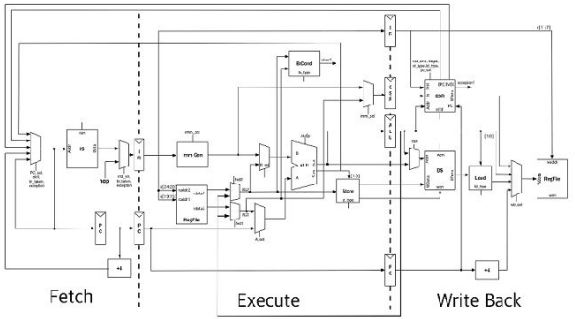
Security Properties:

- G(GPR0==0)
- esr==sr U rst

Tested on 3 Processors







OR1200 mor1kx RISC-V

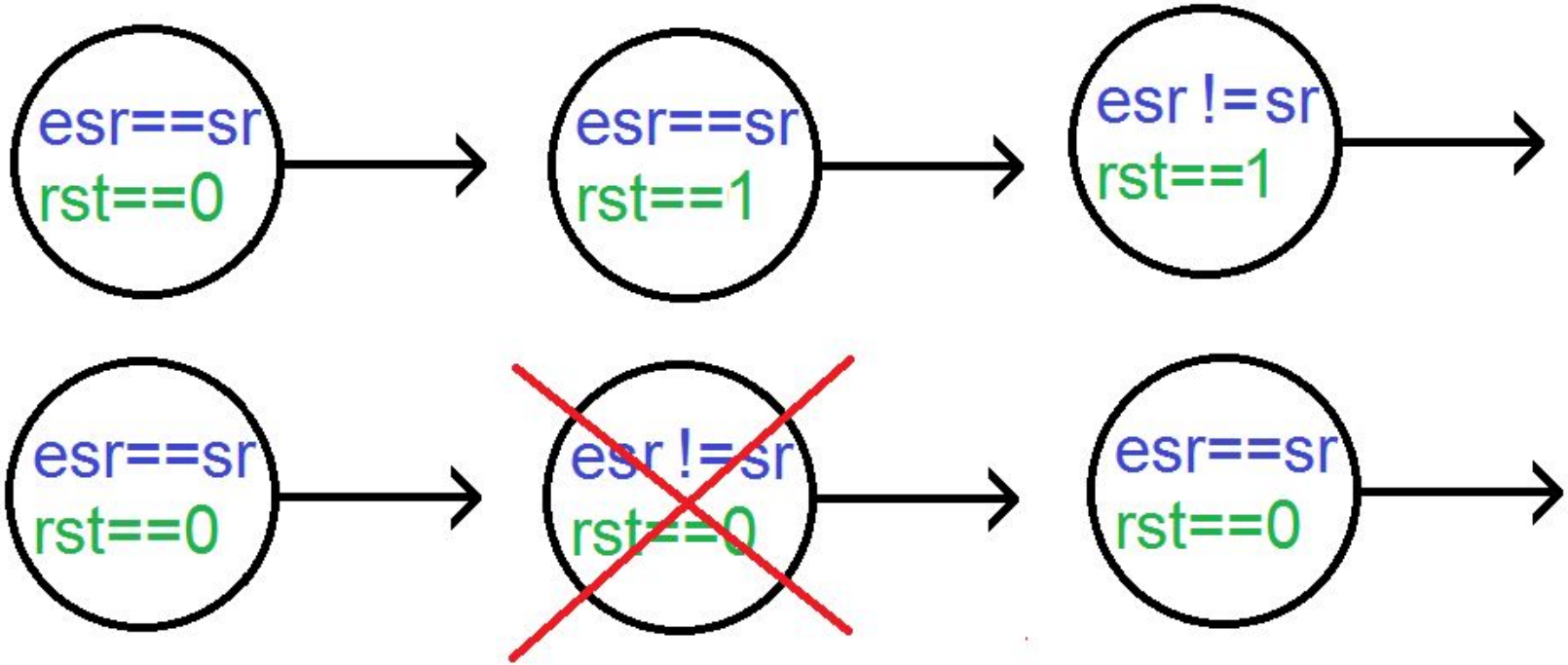
New Temporal Property: sr==esr U G(rst==1)

Discovered and demonstrated a temporal vulnerability to initialization on mor1kx

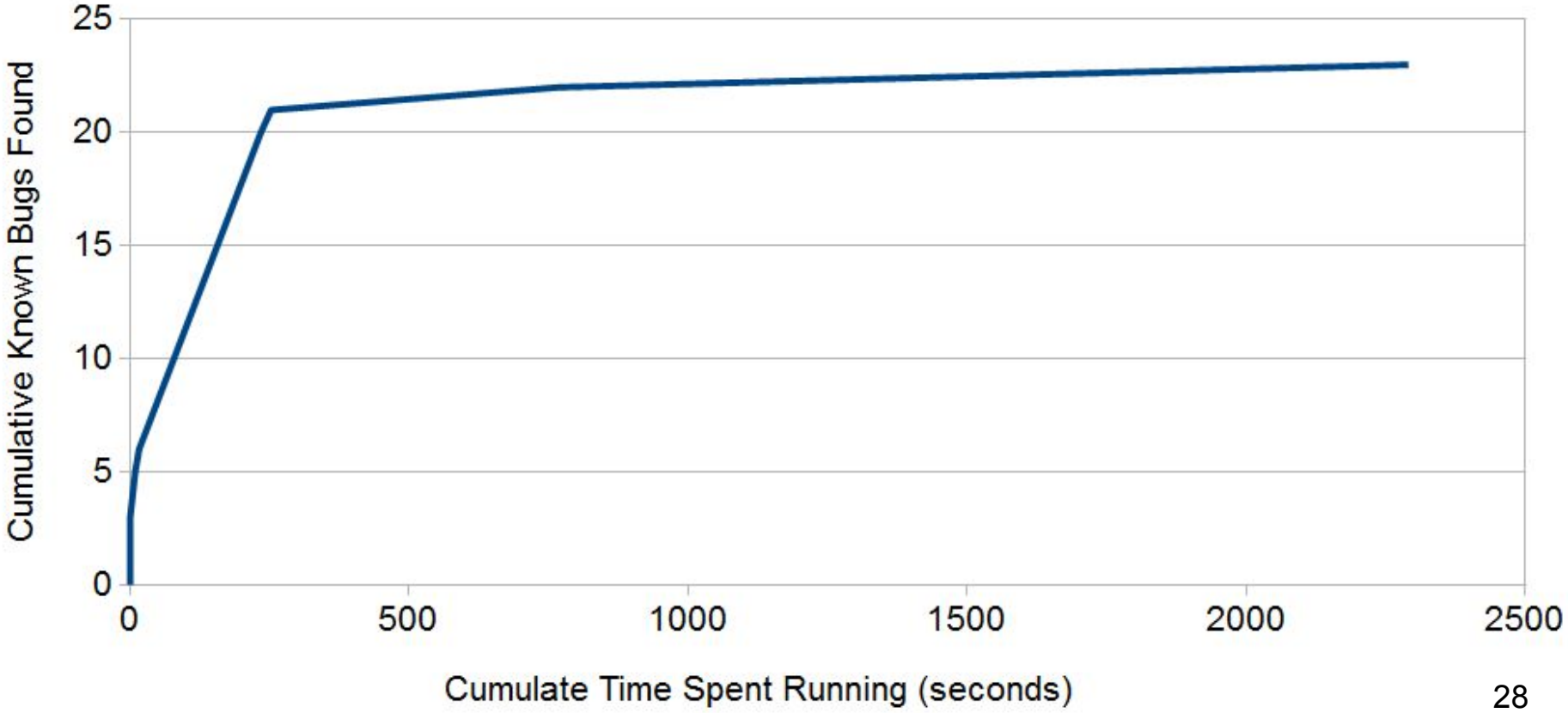
- By default, exception status register (esr) equals status register (sr) on boot.
- However, we can insert a bug that changes the supervisor bit in esr.
- On return from an exception, privilege may remain elevated as sr loads esr.

Finding initialization vulnerabilities is a major contribution of Undine!

New Temporal Vulnerability: sr==esr U G(rst==1)



Found 23 of 29 known bugs - others outside of types



Undine: Mining LTL Properties on RISC

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

Thesis

Specification mining can discover

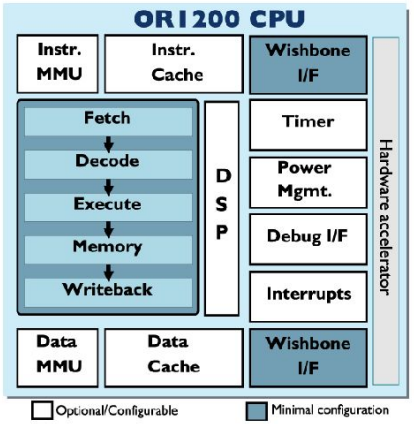
- ✓ **linear temporal logic** security properties such as those related to correct initialization of a system,
- security properties preconditioned on control signals in **closed source CISC designs**, and
- security **hyperproperties** related to information flow.

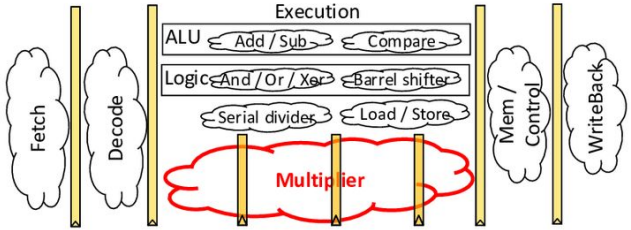
Astarte: Mining Closed Source CISC

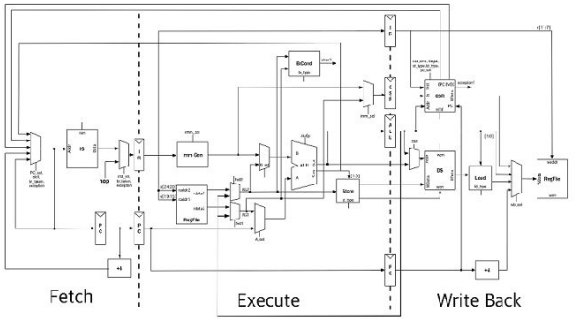
How can properties that model secure behavior of **closed source CISC** designs be discovered using specification mining?

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







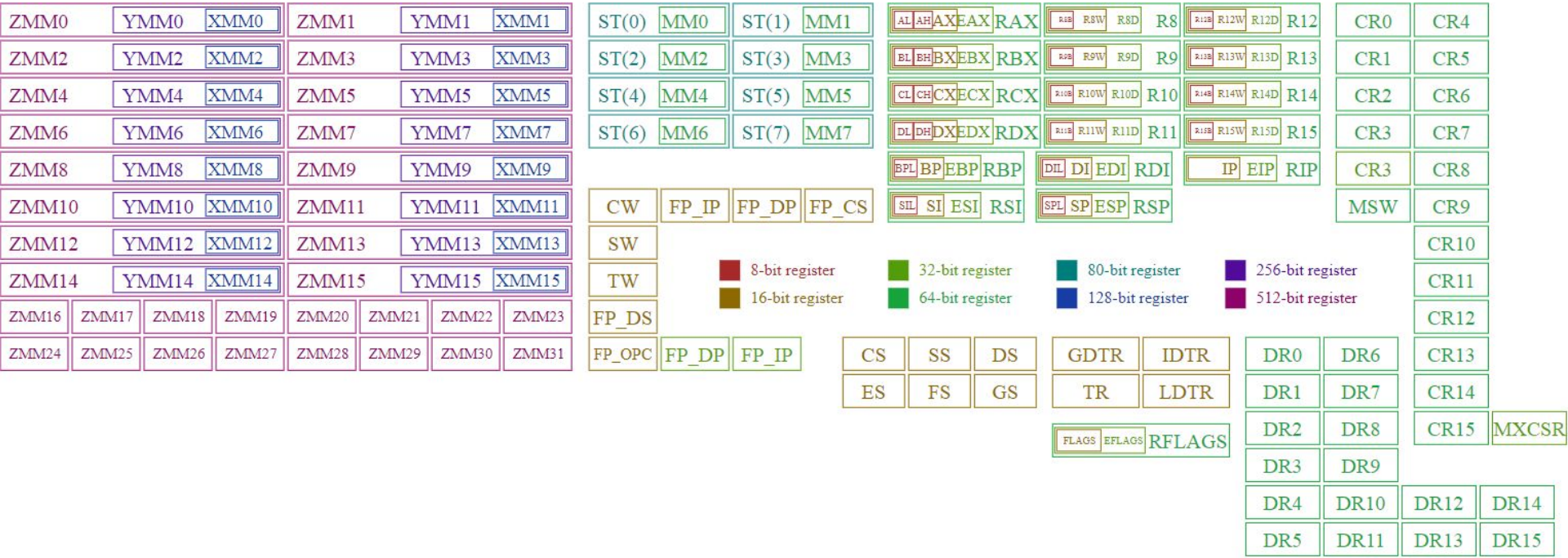
OR1200 mor1kx RISC-V

Recall: Undine Tested on 3 Processors

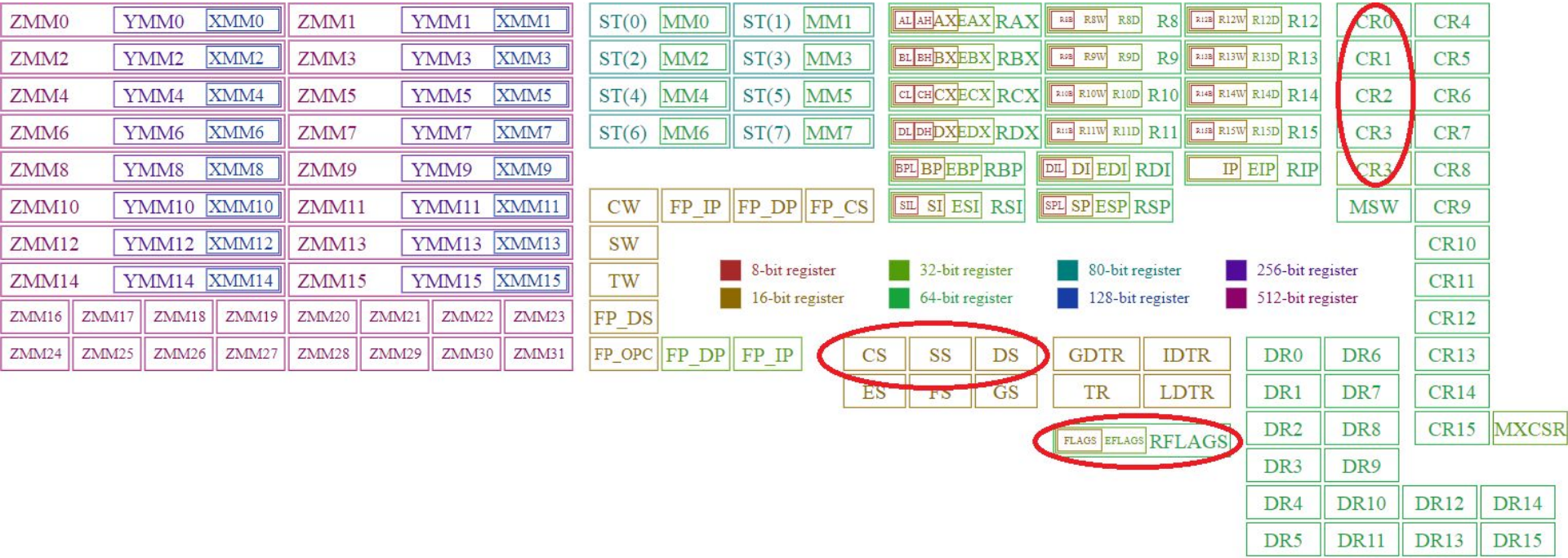
For all 3:

- RTL design
- Access to bug trackers and known bugs (used in SCIFinder to find properties)
- Designs are RISC

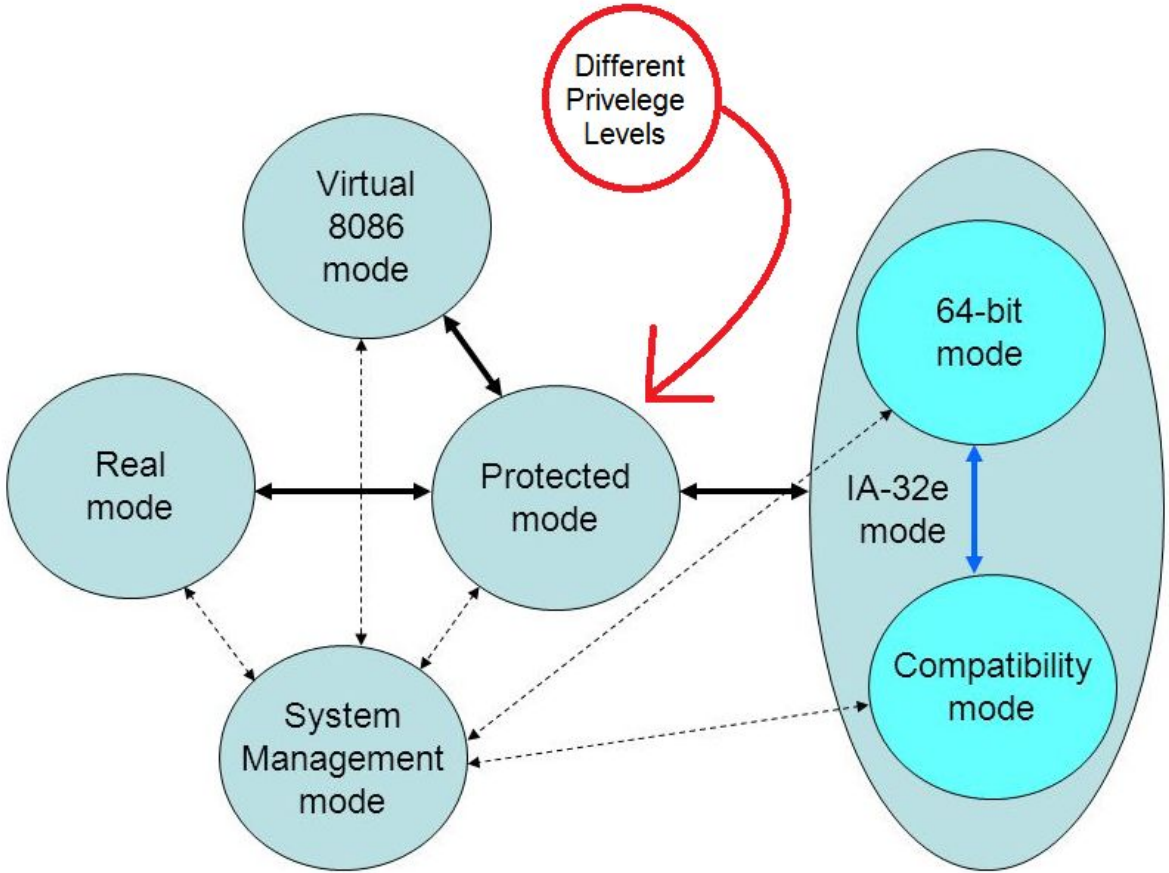
All were Open Source and RISC! x86 is neither!



The x86 specification has many control signals...



Control signals give different secure behaviors!



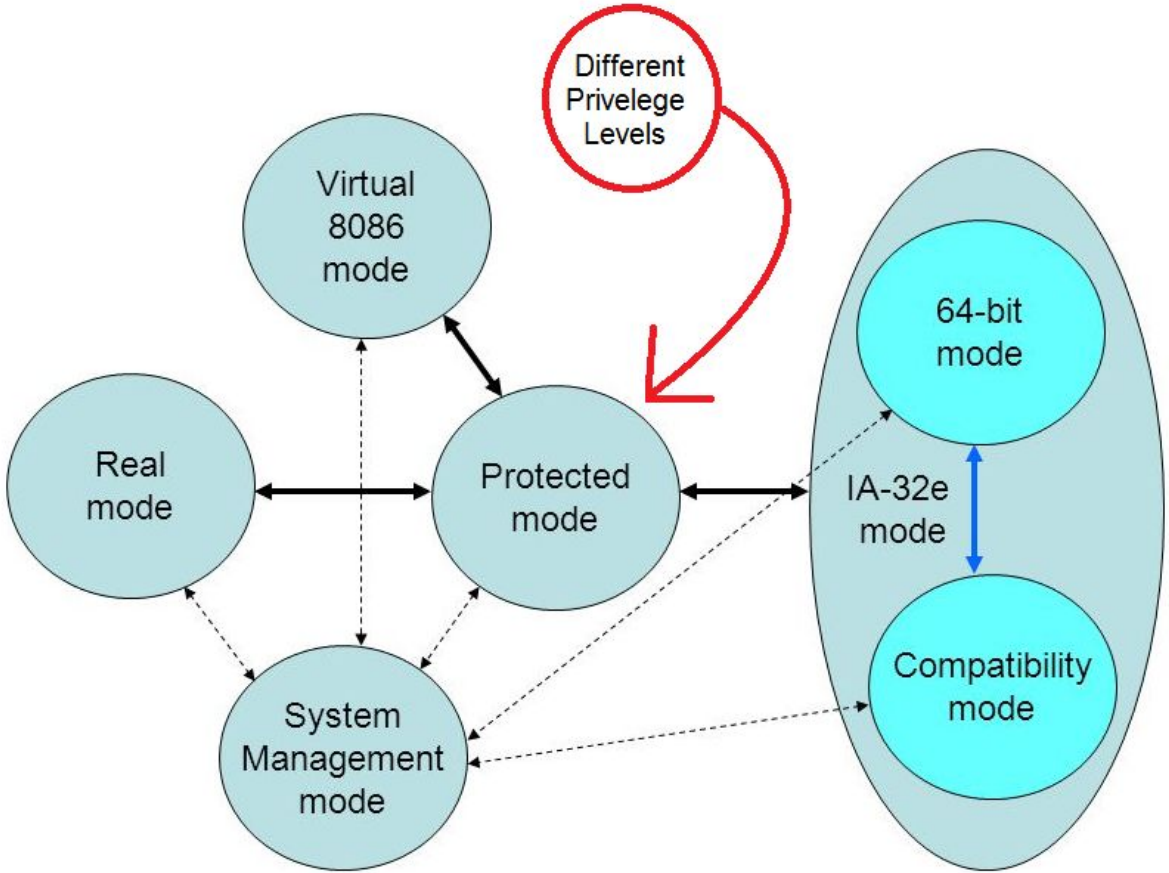
Control signals give different secure behaviors!

For example, the IOPL (I/O privilege level) signal can only be changed at “Ring 0” in protected mode, that is

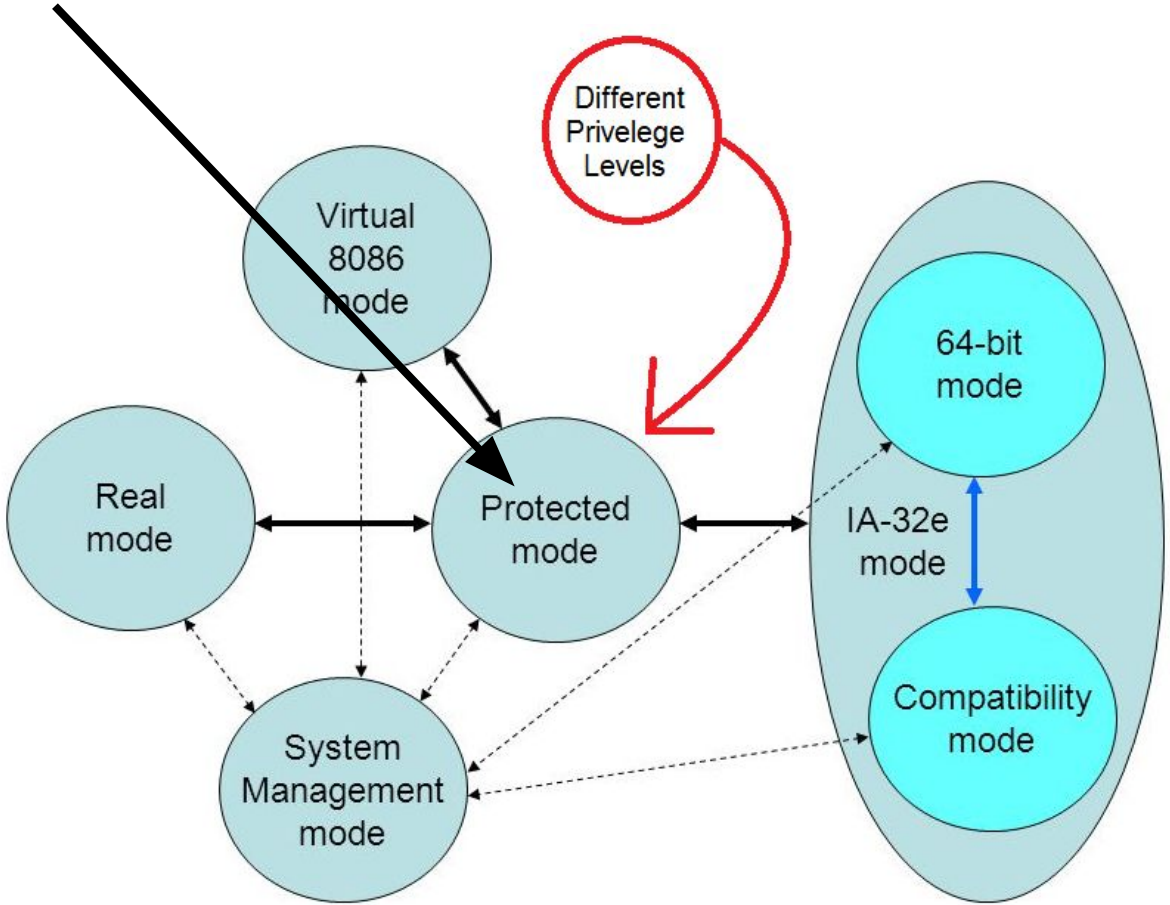
!CPL=0 & PME=1 -> IOPL=orig(IOPL)

“Current privilege level not zero and in protected mode means IOPL can’t change”

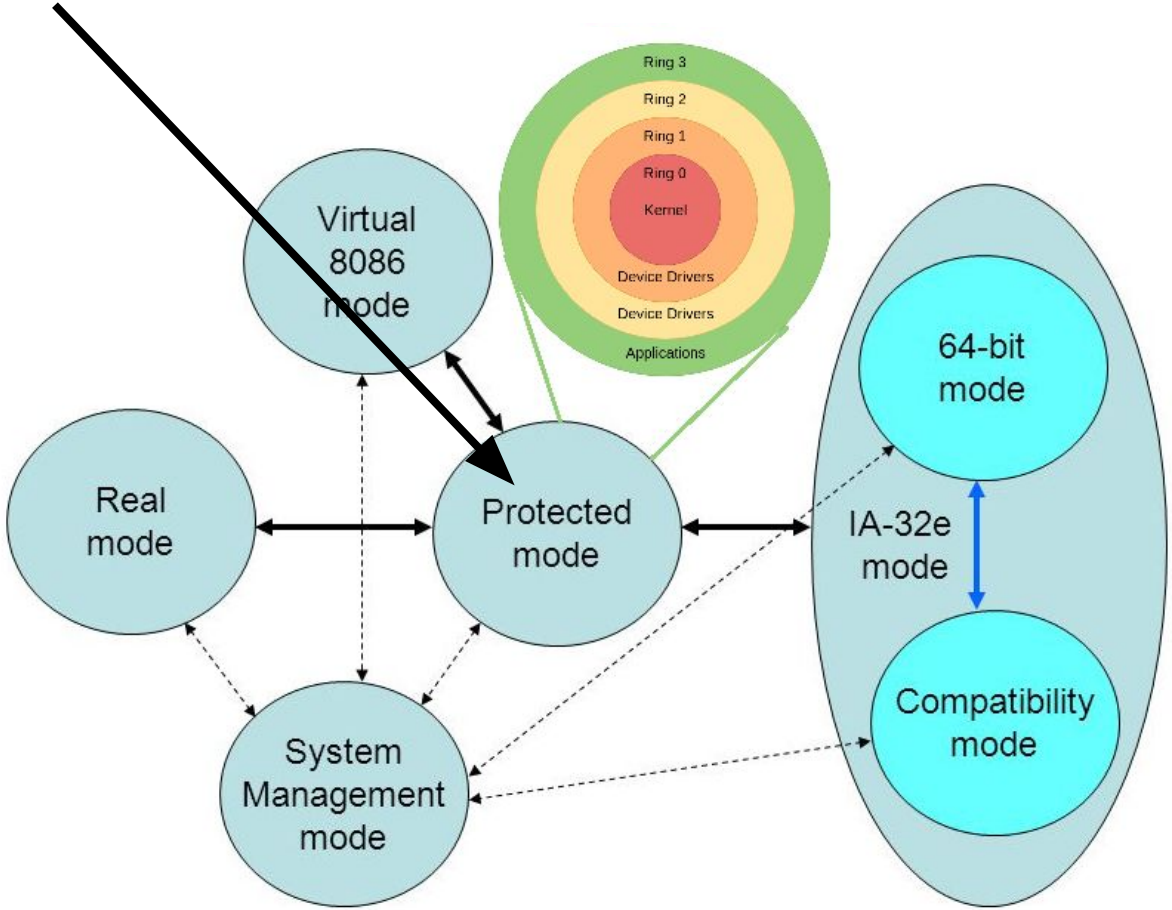
!CPL==0 & PME==1 ==> IOPL==orig(IOPL)



!CPL==0 & **PME==1** ==> IOPL==orig(IOPL)



!CPL==0 & **PME==1** ==> IOPL==orig(IOPL)



!CPL==0 & PME==1 ==> IOPL==orig(IOPL)

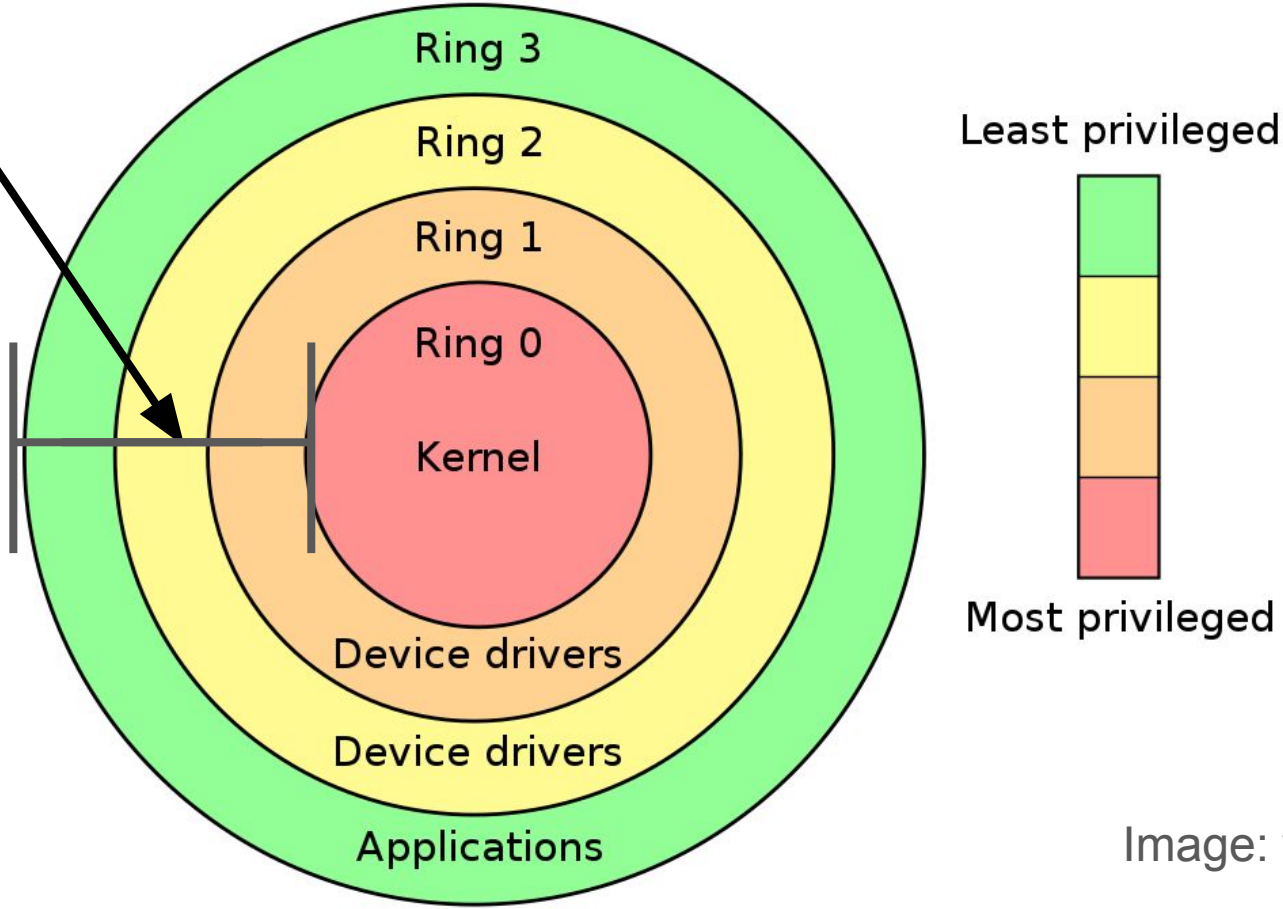
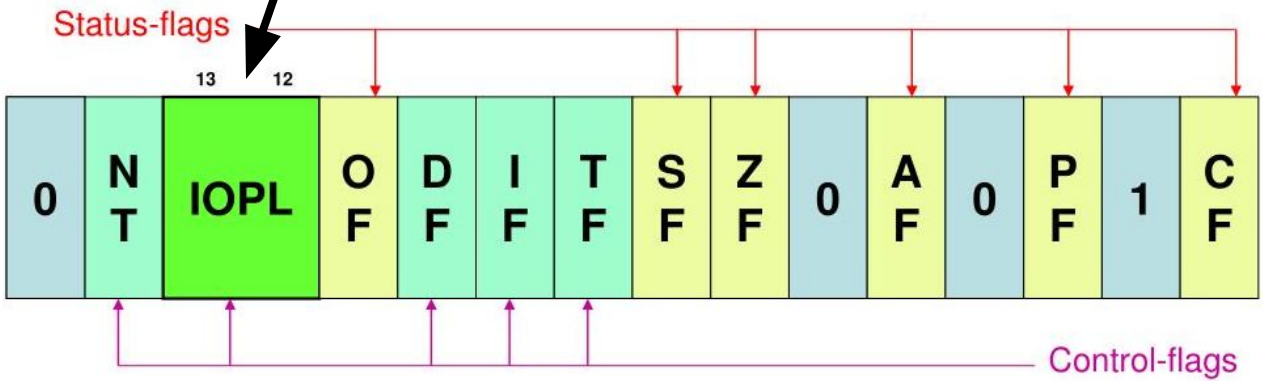


Image: wikimedia

!CPL==0 & PME==1 ==> **IOPL==orig(IOPL)**



x86 EFLAGS register

Why look at control signals?

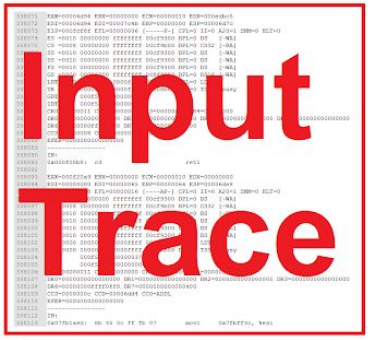
The x86 architecture is large and complex, but can be mined effectively.

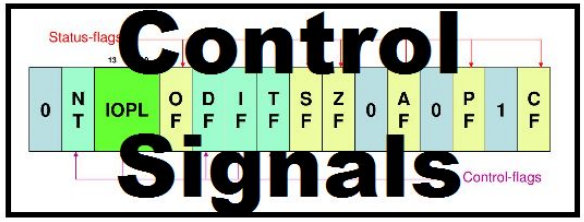
- Control signals implement secure computing.
- Control signals can behave differently across architectures.
- Control signals refine the search.

These control signals can be discovered without the manual.

This replaces our need for known bugs from Undine and increases automation!

So I created a tool to find invariants across signals.





Mining for Secure Behavior

Rather than rely on the manual, I use an automated process.

- I sort software visible signals by type, including into a control register type
- Within control registers, we decompose the register into control bits
- Daikon mines invariants over each control bit

This captures signals in their implemented state without consulting documentation.

Control bits associated with some invariants over the processor are mined further.

Control Signal Examples

Example 1: CR0[4] == 0 in all cases

Per the spec, CR0[4] is a 386-only register and unused on x86

Example 2: CR0[11] is not constant but frequently equal to other control bits

Per the spec, CR4[11] is UMIP, user mode instruction prevention

We find it changes with certain paging values, alignments, modes

Per the spec, it defines whether descriptor tables may be modified at CPL > 0

Control Signals to Preconditions

Control signals may be interesting preconditions in a few ways:

- Signal=*n*, capturing what behavior the signal encodes
- Signal!=orig(Signal), capturing what behavior is allowed to change the signal

Given these preconditions, I pass back over the traces with the preconditions explicitly defined.

Security Control Signals

CS[13]	CPL	Current Privilege Level	Gives Ring in Protected Mode
SMM	SMM	System Management Mode	Gives “Ring -2” or System Management Mode
EFL[6]	ZF	Zero Flag	Indicates Zero result for Arithmetic
EFL[9]	IF	Interrupt enable Flag	Allows or Disallows Interrupts
EFL[11]	OF	Overflow Flag	Indicates Zero result for Arithmetic
EFL[14]	AF	Adjust Flag	Indicates Carry result for Arithmetic
CR0[0]	PE	Protected mode Enable	Gives whether Protected Mode is active
CR0[1]	MP	Monitor co-processor	Controls (F)WAIT instructions
CS	CS	Code Segment	Holds current code segment pointer
SS	SS	Stack Segment	Holds current stack segment pointer
DS	DS	Data Segment	Holds current data segment pointer

Evaluation

Developed and ran Astarte over IvyBridge x86 with bare metal and OS traces.

Considered output properties versus manual and historical bugs.

Considered output properties as implemented by various operating systems.

Results

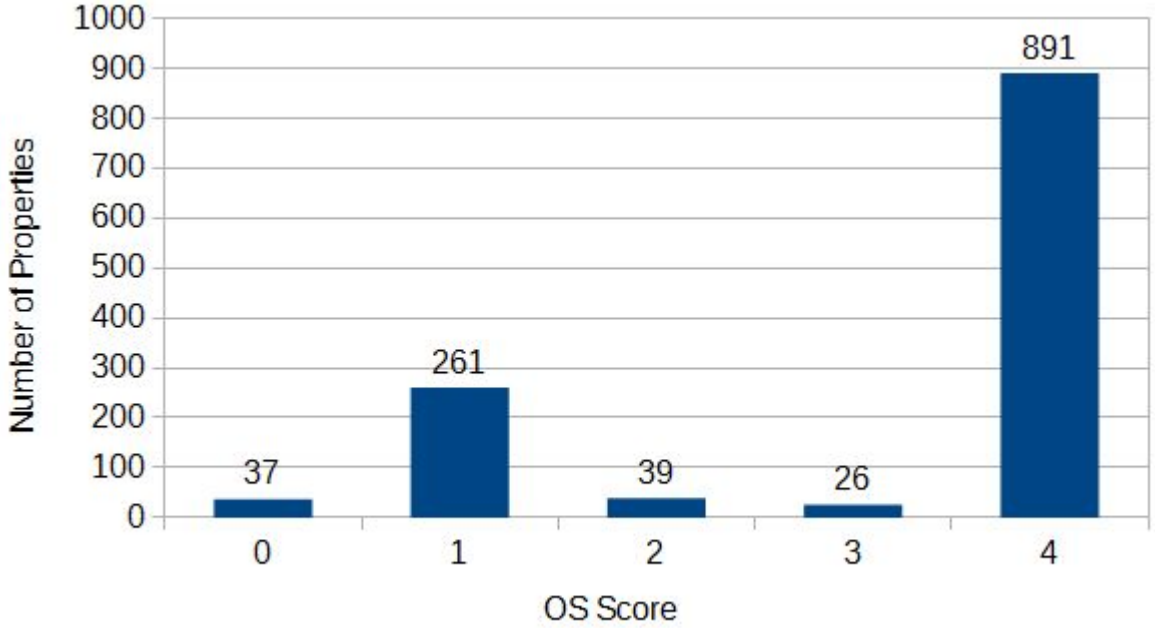
1400 properties

- Found properties for 23 of 29 security properties found in spec, trace limited.
- Found properties for 2 of 2 historical bugs from hardware designs for x86.

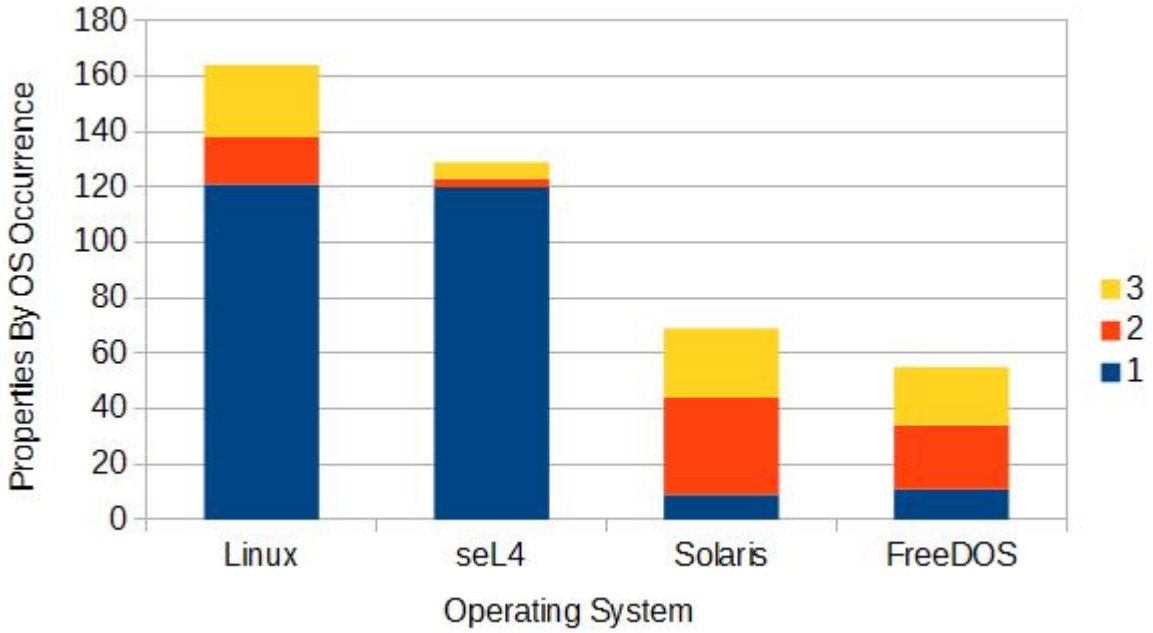
Related properties to Operating Systems

- 900 properties from all operating systems.
- 2 operating systems had 120 unique properties implementing syscalls.
- This gives properties of the design versus OS properties.

Most properties occured in one or all OSes



Most OS-specific properties related to syscalls



Astarte: Mining Closed Source CISC

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

Thesis

Specification mining can discover

- ✓ **linear temporal logic** security properties such as those related to correct initialization of a system,
- ✓ security properties preconditioned on control signals in **closed source CISC designs**, and
- ● security **hyperproperties** related to information flow.

Isadora: Mining Hyperproperties (Proposed work)

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

Hyperproperties

Trace properties are sets of traces. They admit individual traces within the set.

Hyperproperties are sets of properties, or sets of sets of traces.

They admit systems for which all possible traces are one of the sets of traces within the set of sets.

Hyperproperties Example

A property could be:

G (GPR0 == 0)

That is, general purpose register zero always (globally) contains a zero

This property can be determined over a single trace.

Observational Determinism (*OD*) is a hyperproperty

User-level traces appear deterministic regardless of supervisor-level actions.

It takes multiple traces to find if a system adheres to this property as it requires:

- The user-level actions to remain unchanged (that is, appear deterministic).
- The supervisor-level actions to change (that is, be shown as unrestricted).

Formally, with M memory, U user-level memory, and e(M) -> M’ execution:

$$\forall M_1, M_2 : \ (U_1 = U_2) \wedge (e(M_1) \rightarrow^* M_1') \wedge (e(M_2) \rightarrow^* M_2') \Rightarrow (U_1' = U_2')$$

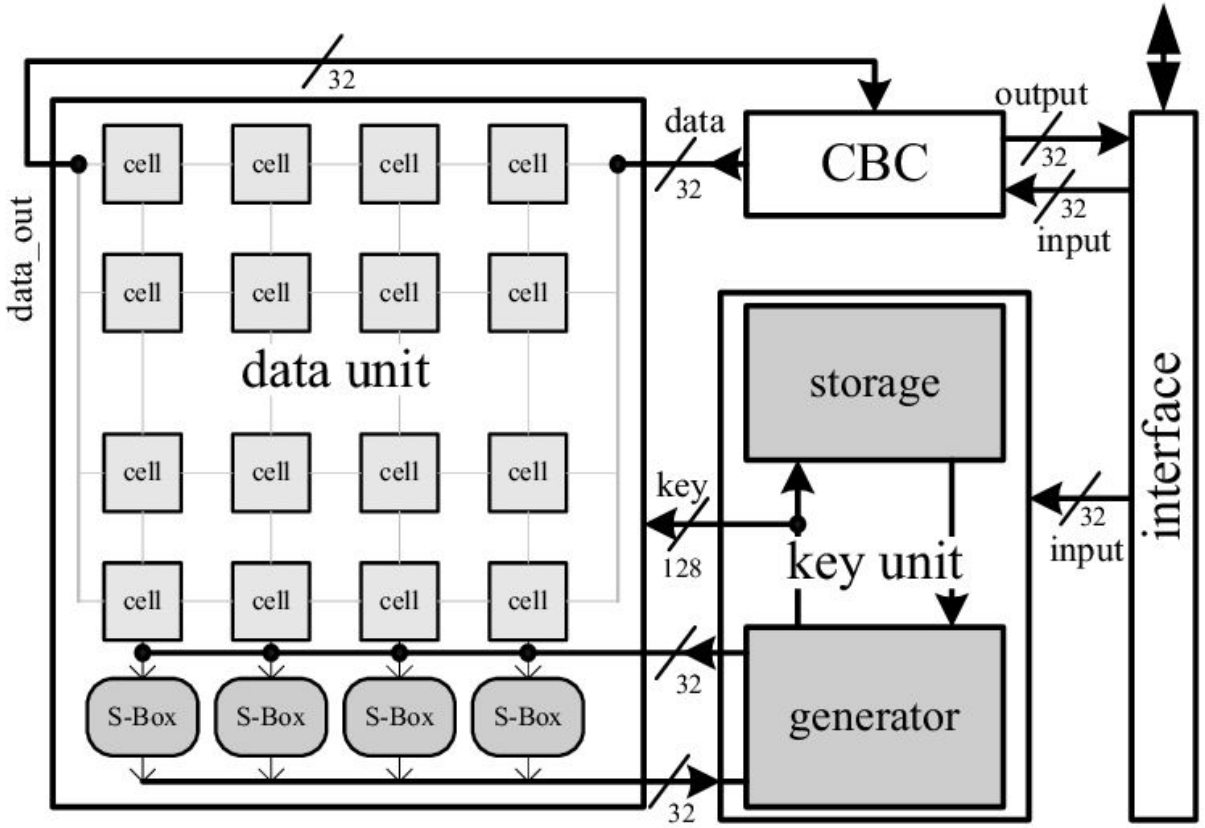
Instrumentation

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

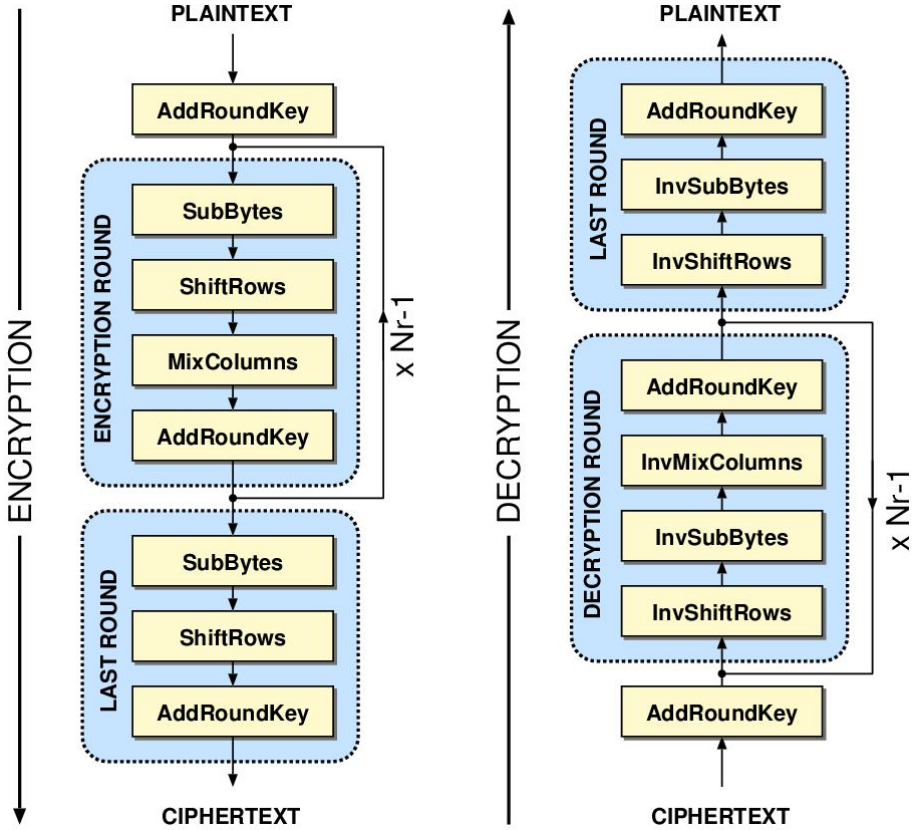
- IFT was developed for software, but works at gate level in simulated designs
- IFT tracks data flows within hardware that originate from certain sources
- *OD* is an information flow hyperproperty

I will collaborate with the Kastner group which has recent work on IFT (Hu 2016).

IFT Example: AES



IFT Example: AES



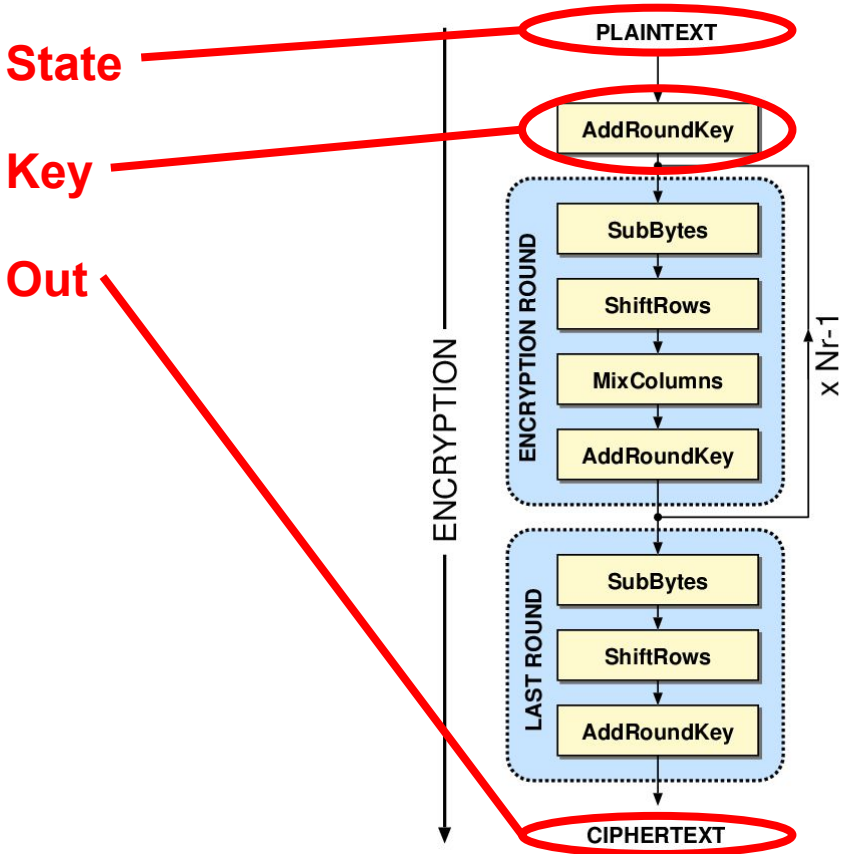
IFT Example: AES

Consider an AES module. It has three main registers:

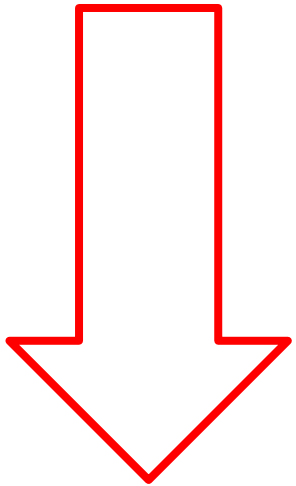
- State
- Key
- Out

Want to ensure that Key and State are both flowing to Out.

IFT Example: AES

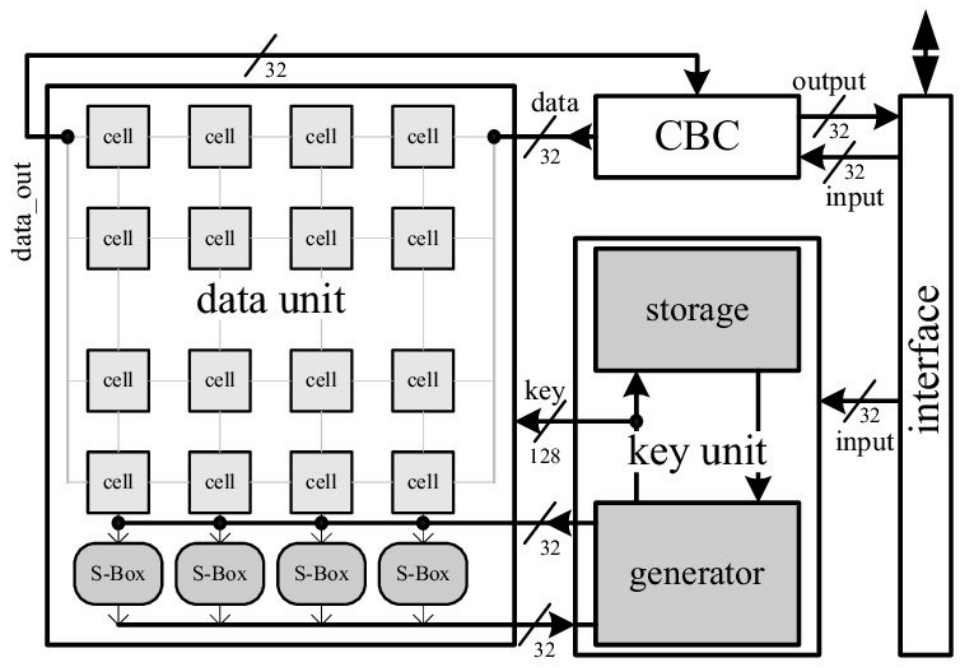


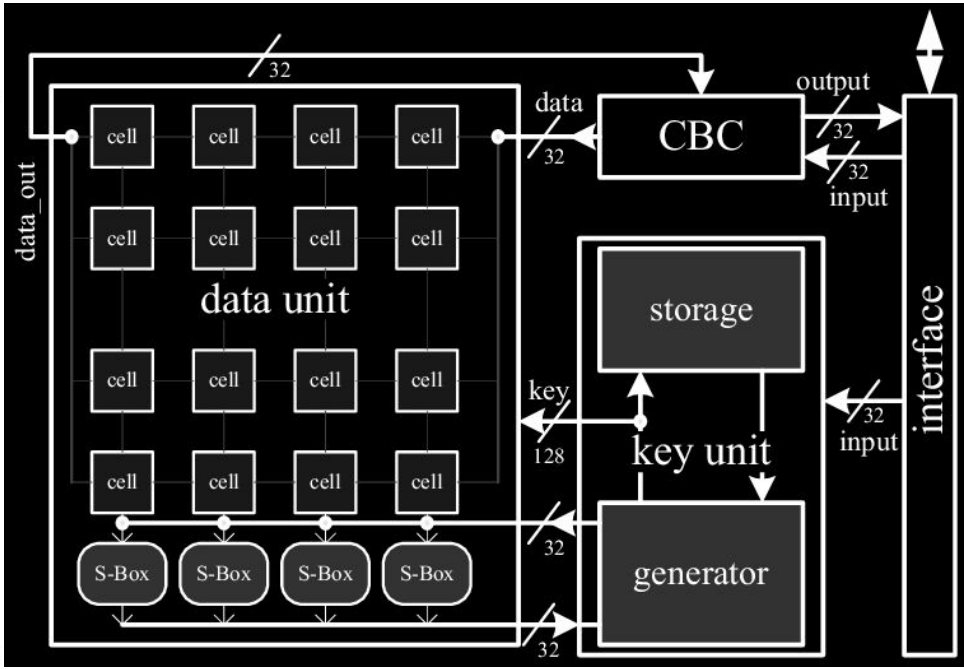
Information flows down through graphic.



IFT Example: AES

Create shadow state of the module for tracking information flow.





IFT Example: AES

Add new tracking elements:

- State_t
- Key_t
- Out_t

Can track flow through elements of processor state using these shadow elements.

Can add these elements to a design a generate traces from design simulation.

IFT Example: AES

Information flow hyperproperties over the original design correspond to trace properties over the IFT instrumented design.

Recall: “We want to ensure that Key and State are both flowing to Out”

- Key_t -> Out_t
- State_t -> Out_t
- !Key_t & !State_t -> !Out_t

Should find no relation between tracking registers and original registers.

Research Goals

I need:

- Trace properties from IFT instrumented traces over different architectures.

I have:

- An IFT instrumented trace as a value change dump.
  - This is the AES example used here.
- Scripts to convert value change dumps to traces for a specification miner.
  - The VCD scripts are for Texada, but likely want to change them for Daikon.
- Output filters for specification miners.
  - These are not configured for hyperproperties necessarily.

Research Plan

Develop and demonstrate hyperproperty mining using IFT traces over AES.

Apply techniques to additional modules, including RSA.

Apply techniques to side channel attacks, such as the data confidentiality of the Common Evaluation Platform (CEP) System on a Chip design.

Evaluation

After the tool is developed, I will evaluate its ability to find information flow security hyperproperties on RISC-V systems that unlike CEP do not have security targets.

I will assess the tool’s ability to prevent side channels using IFT mining.

Thesis

Specification mining can discover

✓ **linear temporal logic** security properties such as those related to correct initialization of a system,

✓ security properties preconditioned on control signals in **closed source CISC designs**, and

❒ security **hyperproperties** related to information flow.

Timeline

Proposal: Fall 2019

Orals: Spring 2020

Isadora: Fall 2020

Writing: Spring 2021

Defense: Spring/Summer 2021

Citations

Identifying Security Critical Properties for the Dynamic Verification of a Processor, Rui Zhang, Natalie Stanley, Christopher Griggs, Andrew Chi, Cynthia Sturton. ASPLOS 2017.

Wei Hu et al., "Imprecise security: Quality and complexity tradeoffs for hardware information flow tracking," 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, 2016, pp. 1-8.

The End

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL