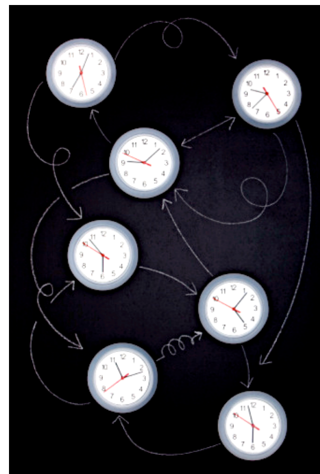
A software-hardware contract for controlling cache and timing channels.

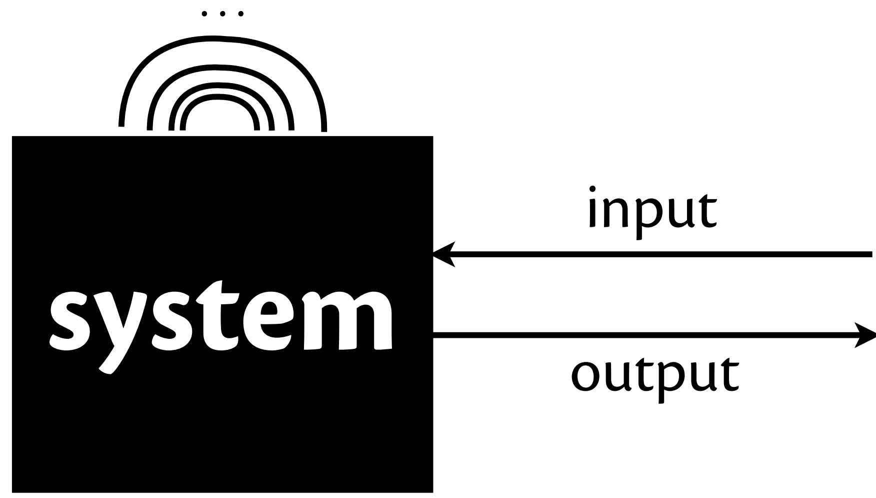
Andrew Myers
Cornell University
(with Danfeng Zhang, Aslan Askarov)



Side channels:

time
power
EM radiation
...

The adversary can learn (a lot) from side channels.



Timing channels:

Known to exist

Hard to detect

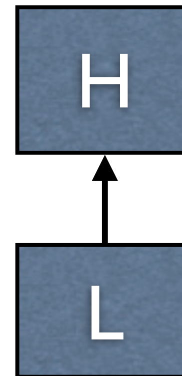
Hard to prevent

A few timing attacks

- Network timing attacks
  - RSA keys leaked by decryption time, measured across network [Brumley&Boneh'05]
  - Load time of web page reveals login status, size and contents of shopping cart [Bortz&Boneh'07]
- Cache timing attacks
  - AES keys leaked by timing memory accesses [Osvik et al'06] from ~300 (!) encryptions. Even works across VMMs. [Zhang et al.'12]
- Covert timing channels
  - Transmit confidential data by controlling response time, e.g., combined with SQL injection [Meer&Slaviero'07]
- Timing channels : a serious threat

Security policies

- Security policy lattice
  - Information has label describing intended confidentiality
  - For this talk, a simple lattice:
    - L=public, H=secret
    - H should not flow to L



- Adversary powers
  - Sees contents of low (L) memory (data/storage channel)
  - Sees timing of updates to low memory (timing channel)

A timing channel

```
if (h)
  sleep(1);
else
  sleep(2);
```

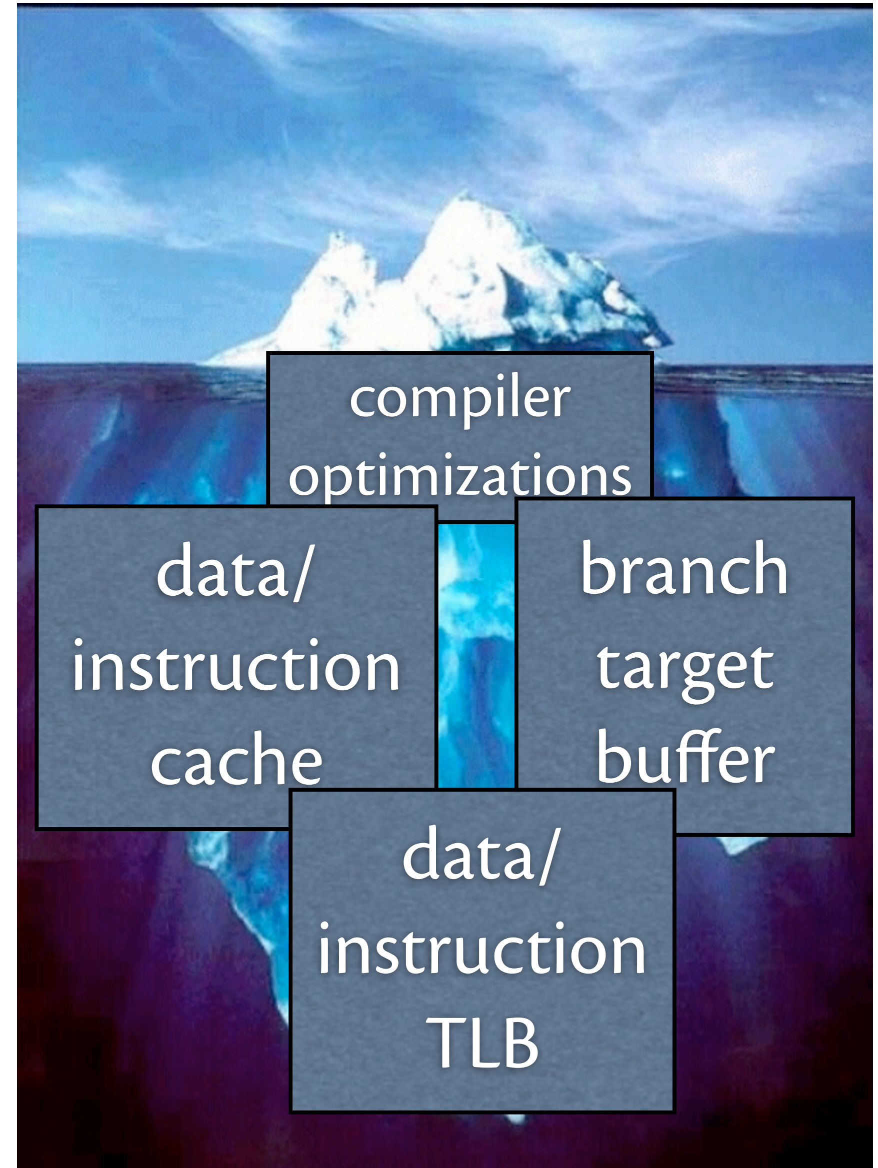
A subtler example

```
if (h1)
  h2=l1;
else
  h2=l2;
l3=l1;
```

- Data cache affects timing!
- Adversary thread can probe l1 to learn h1

Beneath the surface

```
if (h1)
  h2=l1;
else
  h2=l2;
l3=l1;
```



A language-level abstraction [PLDI'12]

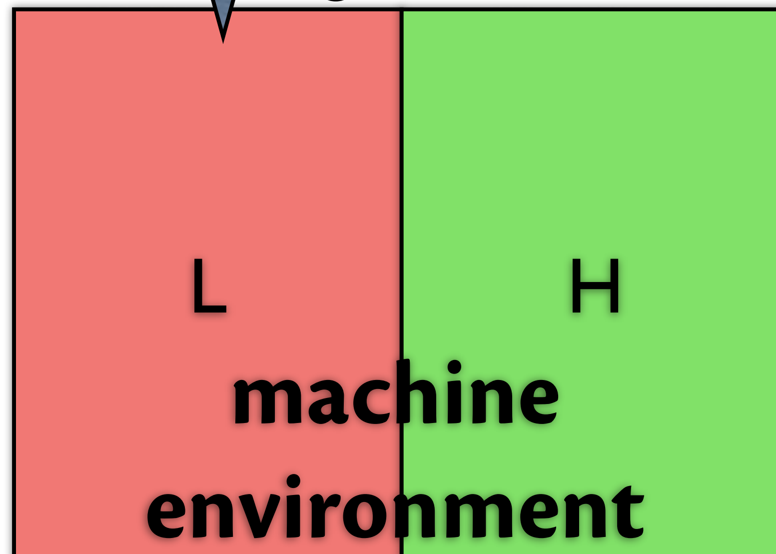
- Each operation has **read label, write label** governing interaction with machine environment

$(x := e)$ $[\ell_r, \ell_w]$

- …nvironment: state affecting …t invisible at language level

seen by
adversary

machine env. logically partitioned by security level (e.g. high cache vs. low cache)



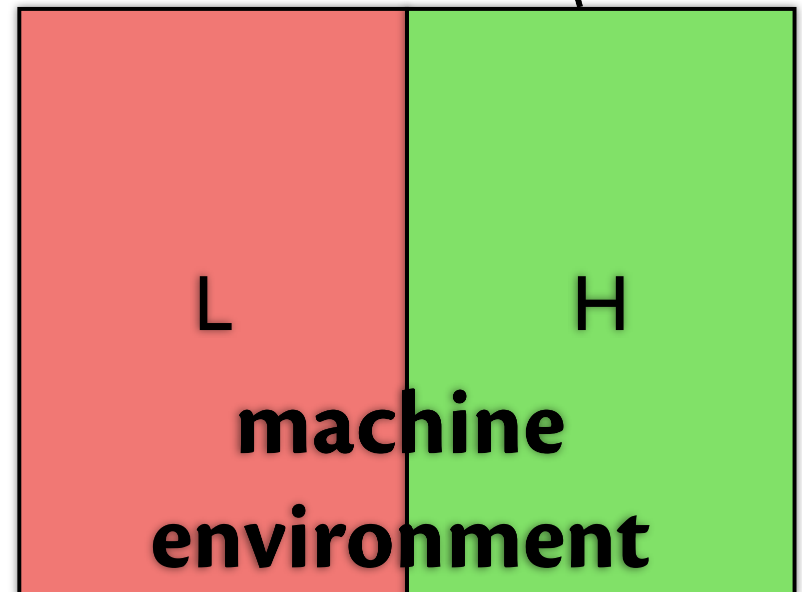
Does *not* include language-visible state (memory)

Read label

$(x := e)_{[\ell_r, \ell_w]}$

abstracts how machine environment affects time taken by next language-level step.

= upper bound on influence



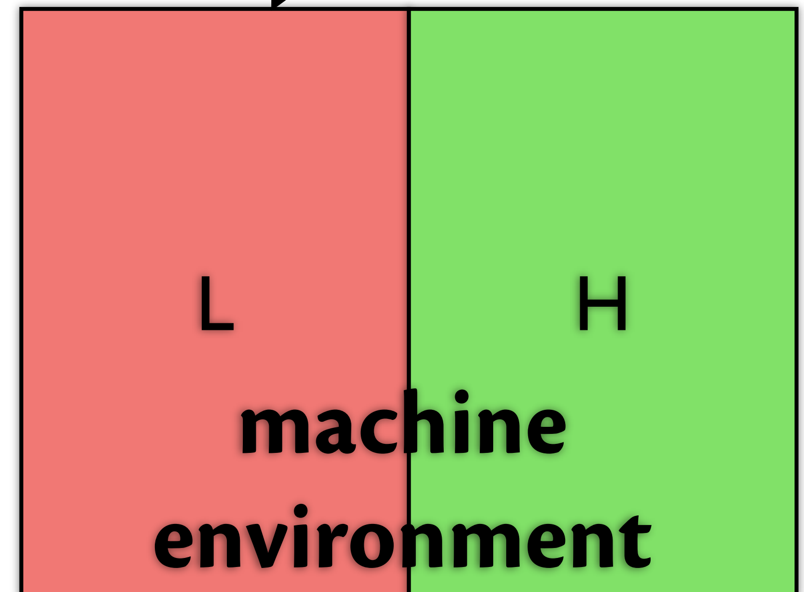
Write label

$(\mathrm{x} := e)_{[\ell_r, \ell_w]}$

$(h_1 := h_2)_{[L,H]}$

abstracts how machine environment is affected by next language-level step

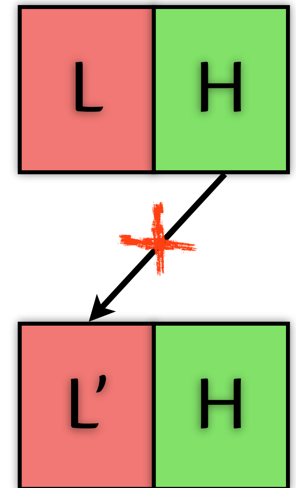
= *lower* bound on effects



The contract

- Language implementation must satisfy three (formally defined) properties:

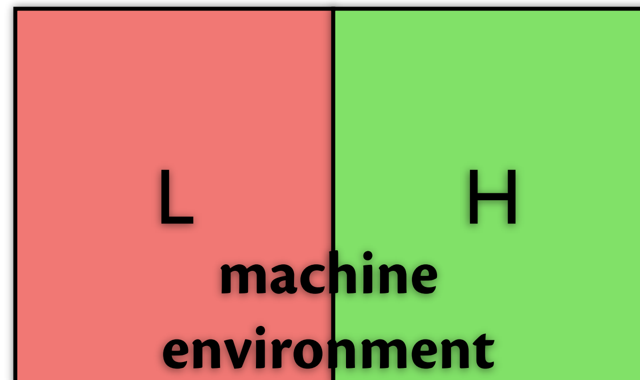
  1. Read label property
  2. Write label property
  3. **Single-step noninterference:** no leaks from high environment to low environment



- Realizable on commodity HW (no-fill mode)
- Provides guidance to designers of future secure architectures

One realization (cache only)

- Cache split into H and L partitions
  - H cache stores lines accessed from high contexts
  - L cache stores lines accessed from low contexts
- Machine environment does not include *data* in cache, only addresses
  - Confidential information can be stored in low cache
  - Low-read-label access to high cache must simulate miss



Type system

- We analyze programs using a type system that tracks timing.

  $c : T$ ⇒ time to run c depends on information of (at most) label T

- A "standard" information flow type system, plus read and write labels.
  - Standard part controls data (storage) channels (e.g., forbids l := h)
  - labels can be generated by inference, optimizer, programmer.

Examples:

$c_{[H,L]} : H$

$(h_1{:=}h_2)_{[L,L]} : L$

$sleep(h) : H$

if (h_1)

$(h_2{:=}l_1)_{[L,H]}$;

else

$(h_2{:=}l_2)_{[L,H]}$;

$(l_3{:=}l_1)_{[L,L]}$

low cache read cannot be affected by h_1

Formal results

- Memory and machine environment noninterference:

A well-typed program* leaks nothing via either internal or external timing channels *or* data channels.

* using no timing mitigation

Language-level mitigation

mitigate(l) { s }

label of running time

mitigated command

- Executes s but adds time using **predictive mitigation** [CCS'10, CCS'11]
  - New expressive power:
    sleep(h) : H *but* mitigate(l) { sleep (h) } : L
- Result: well-typed program using mitigate has bounded leakage (e.g., $O(\log^2 T)$)

Evaluation Setup

- Simulated architecture satisfying security properties with statically partitioned cache and TLB

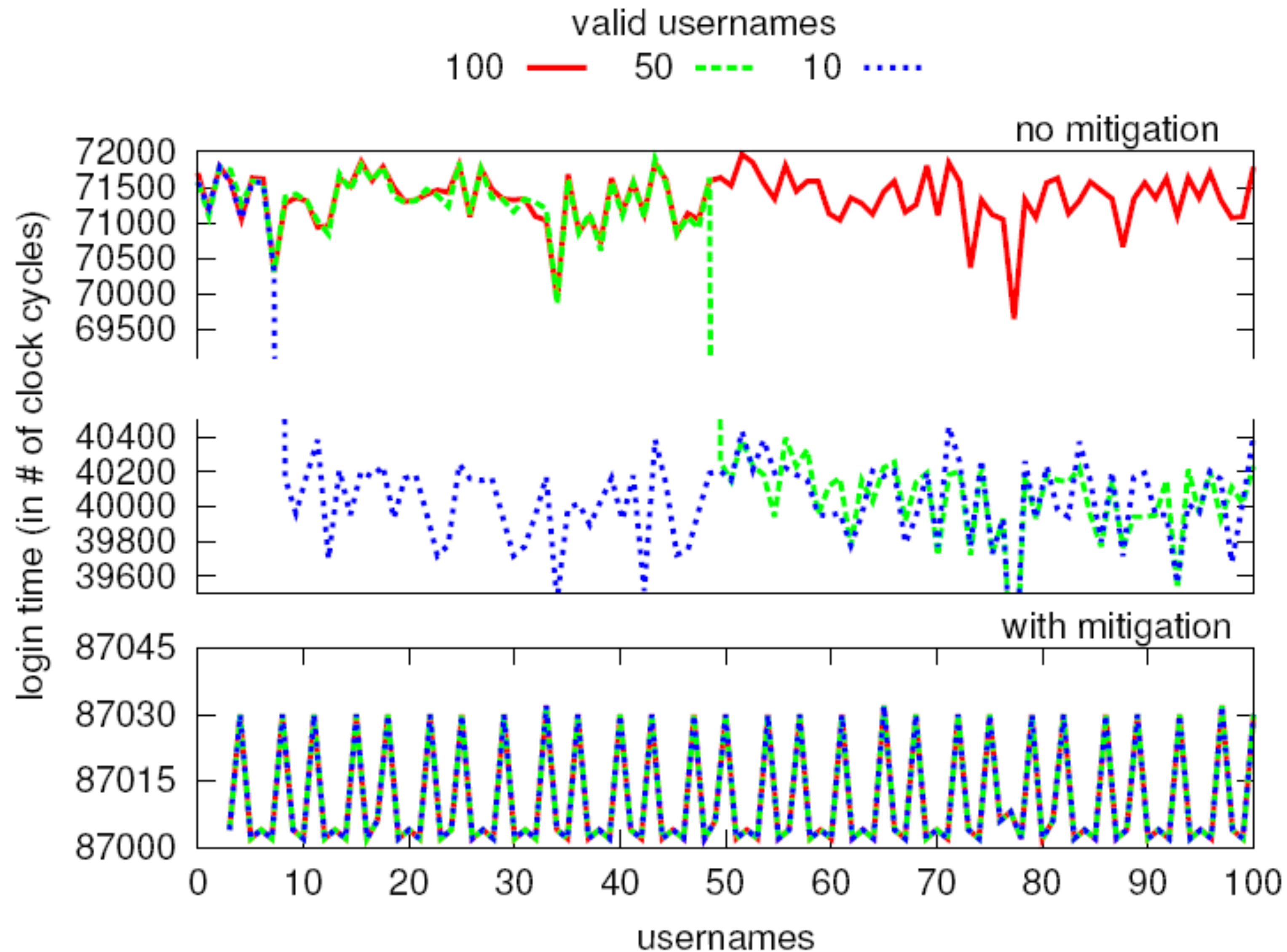
Name	# of sets	issue	block size	latency
L1 Data Cache	128	4-way	32 byte	1 cycle
L2 Data Cache	1024	4-way	64 byte	6 cycles
L1 Inst. Cache	512	1-way	32 byte	1 cycle
L2 Inst. Cache	1024	4-way	64 byte	6 cycles
Data TLB	16	4-way	4KB	30 cycles
Instruction TLB	32	4-way	4KB	30 cycles

- Implemented on SimpleScalar simulator, v.3.0e

Web login example

- Valid usernames can be learned via timing [Bortz&Boneh 07]
- Secret
  - MD5 digest of valid (username, password) pairs
- Inputs
  - 100 different (username, password) pairs

Login behavior



Performance

- nopar: unmodified hardware
- moff: secure hardware, no mitigation
- mon: secure hardware with mitigation

	nopar	moff	mon
ave. time (valid)	70618	78610	86132
ave. time (invalid)	39593	43756	86147
overhead (valid)	1	1.11	1.22

Conclusions

- Timing channels should be reflected at the software/ hardware boundary
  - **Read and write labels** as a clean, general abstraction of hardware timing behavior, enabling software/hardware codesign
  - **Predictive mitigation**, a dynamic mechanism for bounding timing leakage
  - Static analysis of timing behavior with strong guarantees of bounded information leakage.