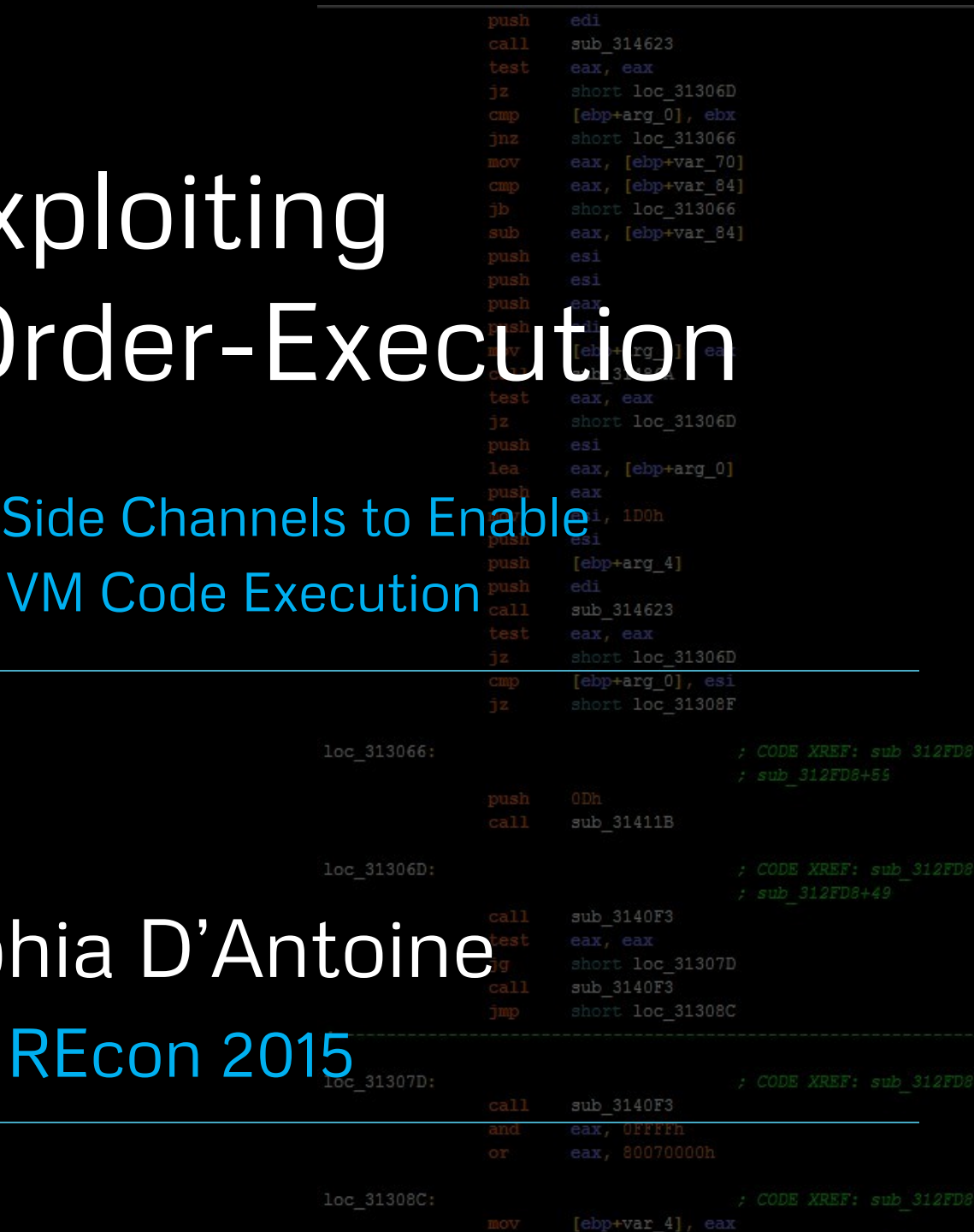
Exploiting Out-of-Order-Execution

Processor Side Channels to Enable Cross VM Code Execution

Sophia D'Antoine

REcon 2015



The Cloud

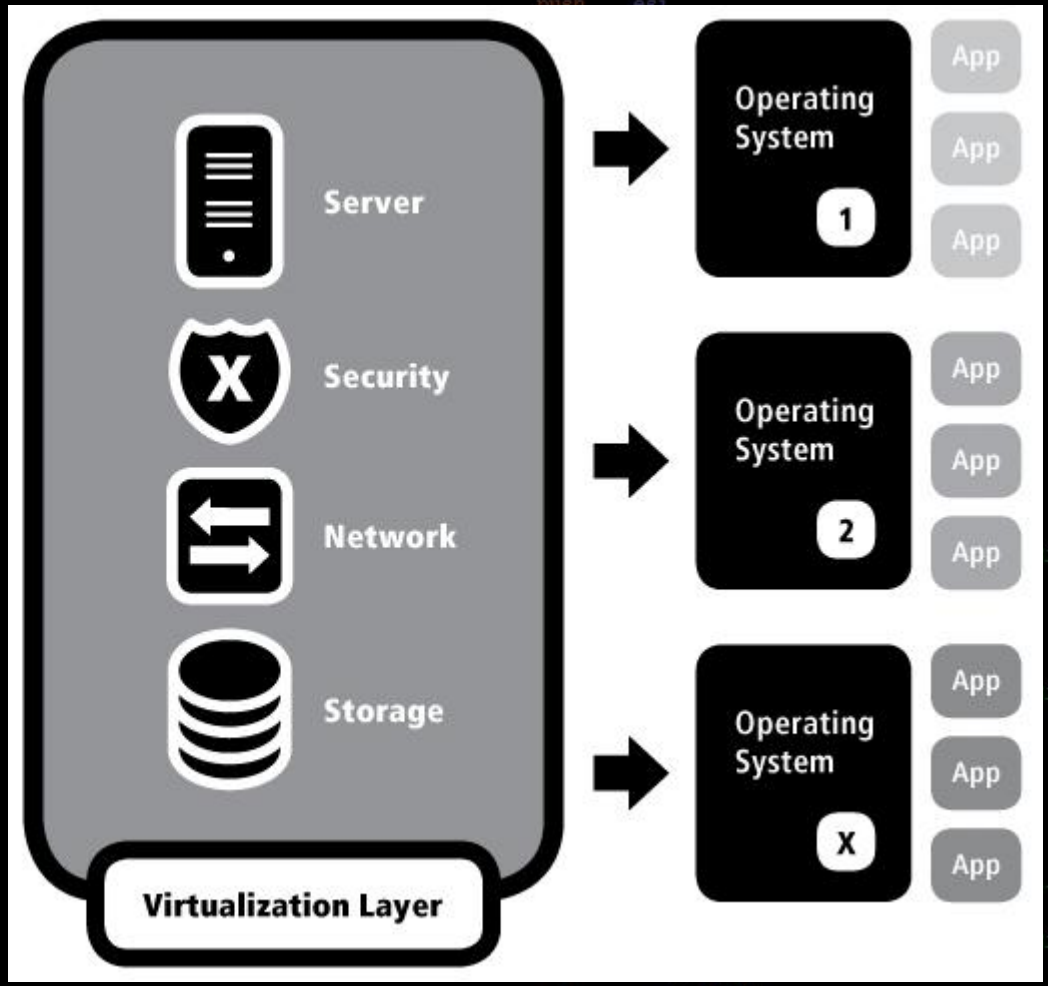

DEMAND A SAFER CLOUD
client-side encryption, server-side computation
beginning with encrypted search
LEARN MORE

Cloud Computing (IaaS)

- Virtual instances
- Hypervisors

Dynamic allocation

=> Reduces cost

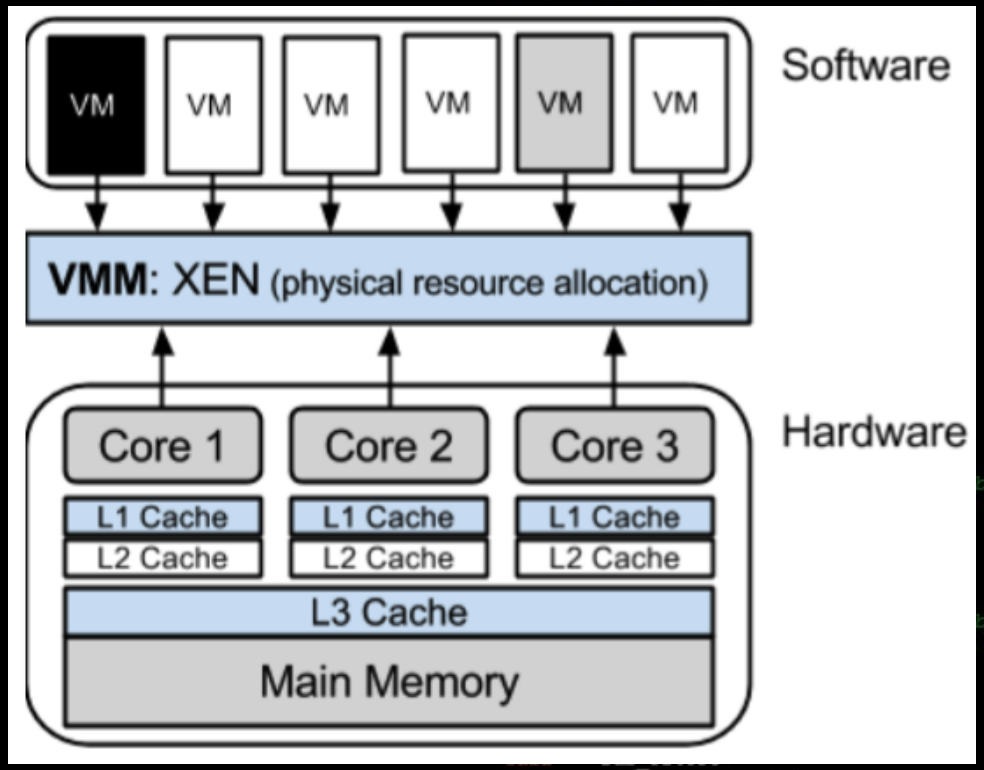


Everyone's Happy

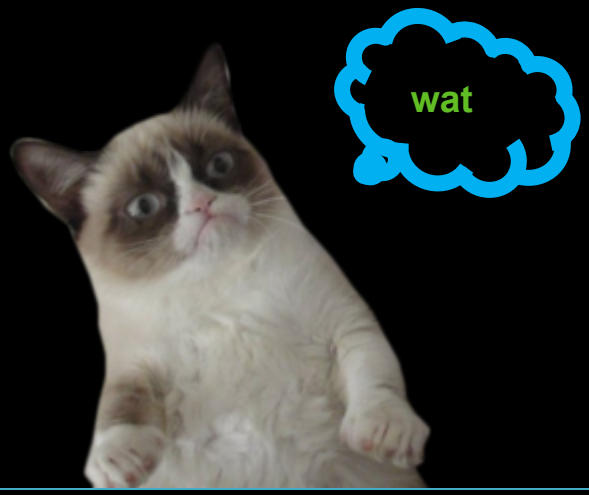
Problems with the Cloud

Security issues with cloud computing

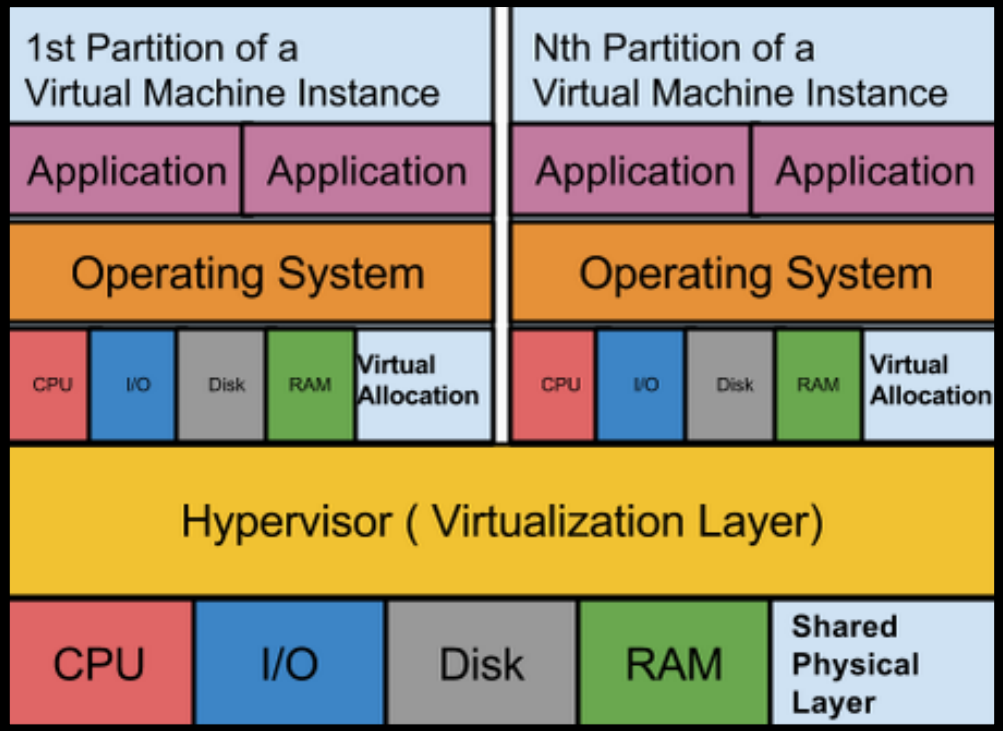
- Sensitive data stored remotely
- Vulnerable host
- Untrusted host
- Co-located with foreign VM's



Physical co-location leads to side channel vulnerabilities.



Cloud Hardware

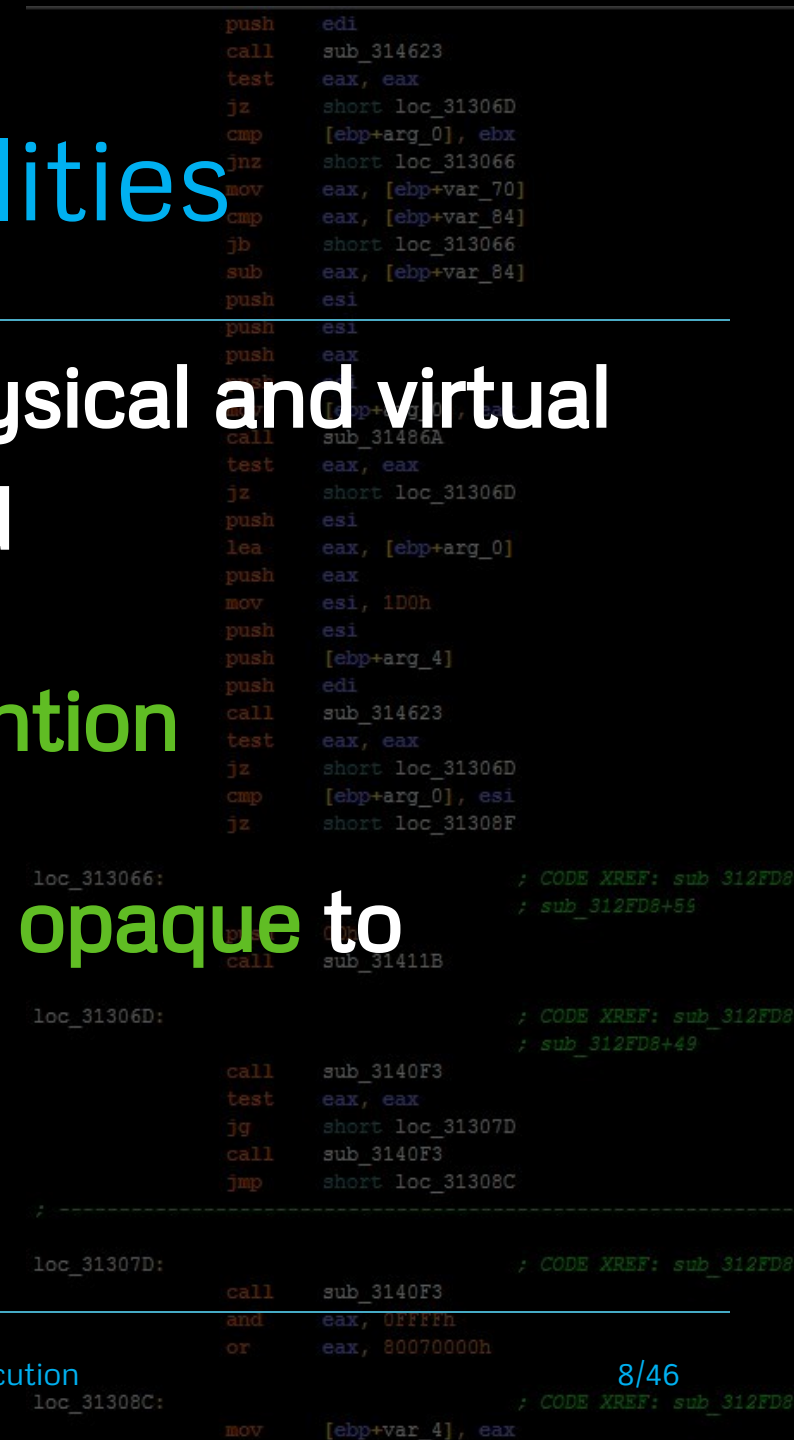


Universal Vulnerabilities

1) Translation between physical and virtual hardware based on need

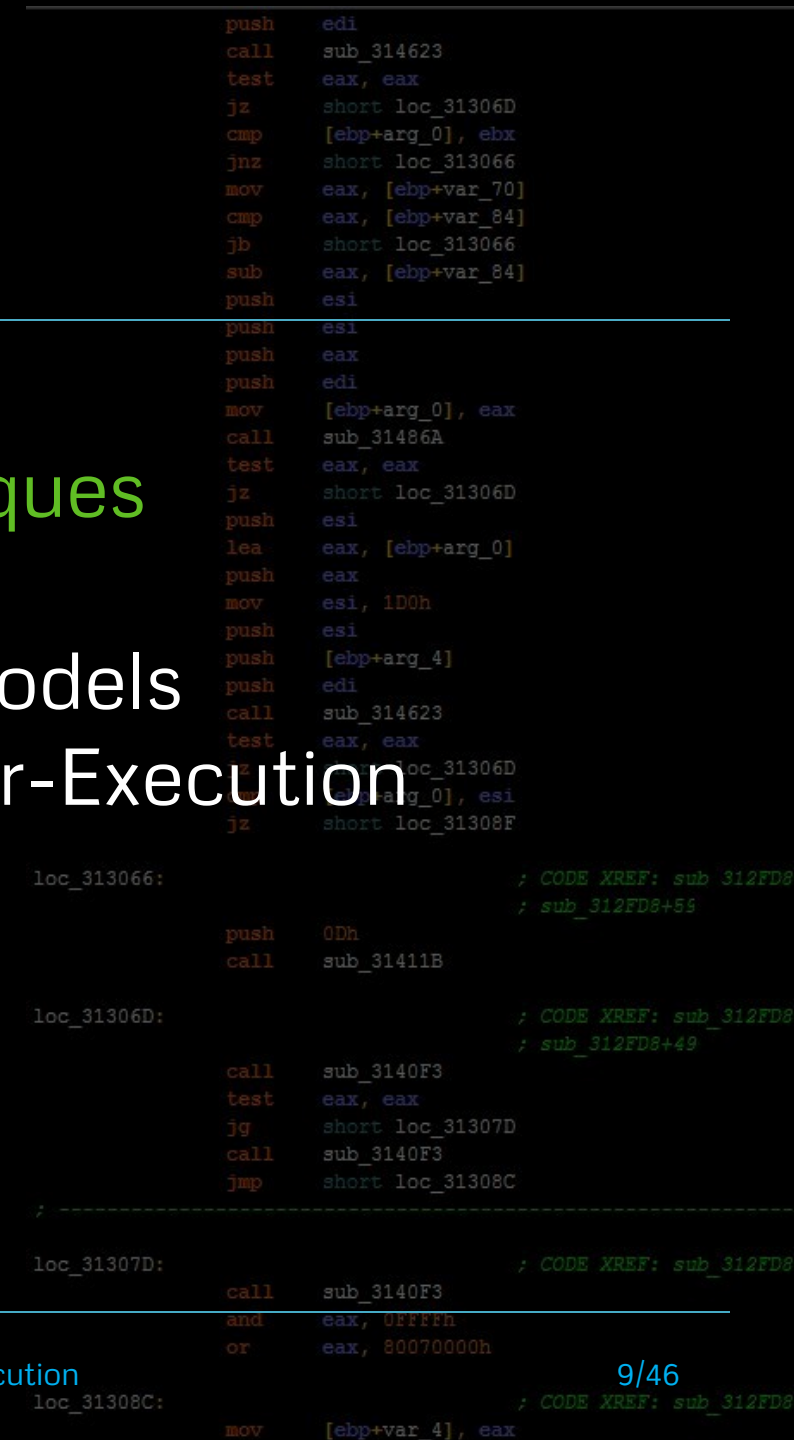
2) Allocation causes contention

3) Private VM activities not opaque to co-residents



Overview

1. Introduction
2. Cloud exploitation techniques
3. Targeting the processor
4. Importance of memory models
5. Design of an Out-of-Order-Execution channel
6. Demo
7. Conclusion

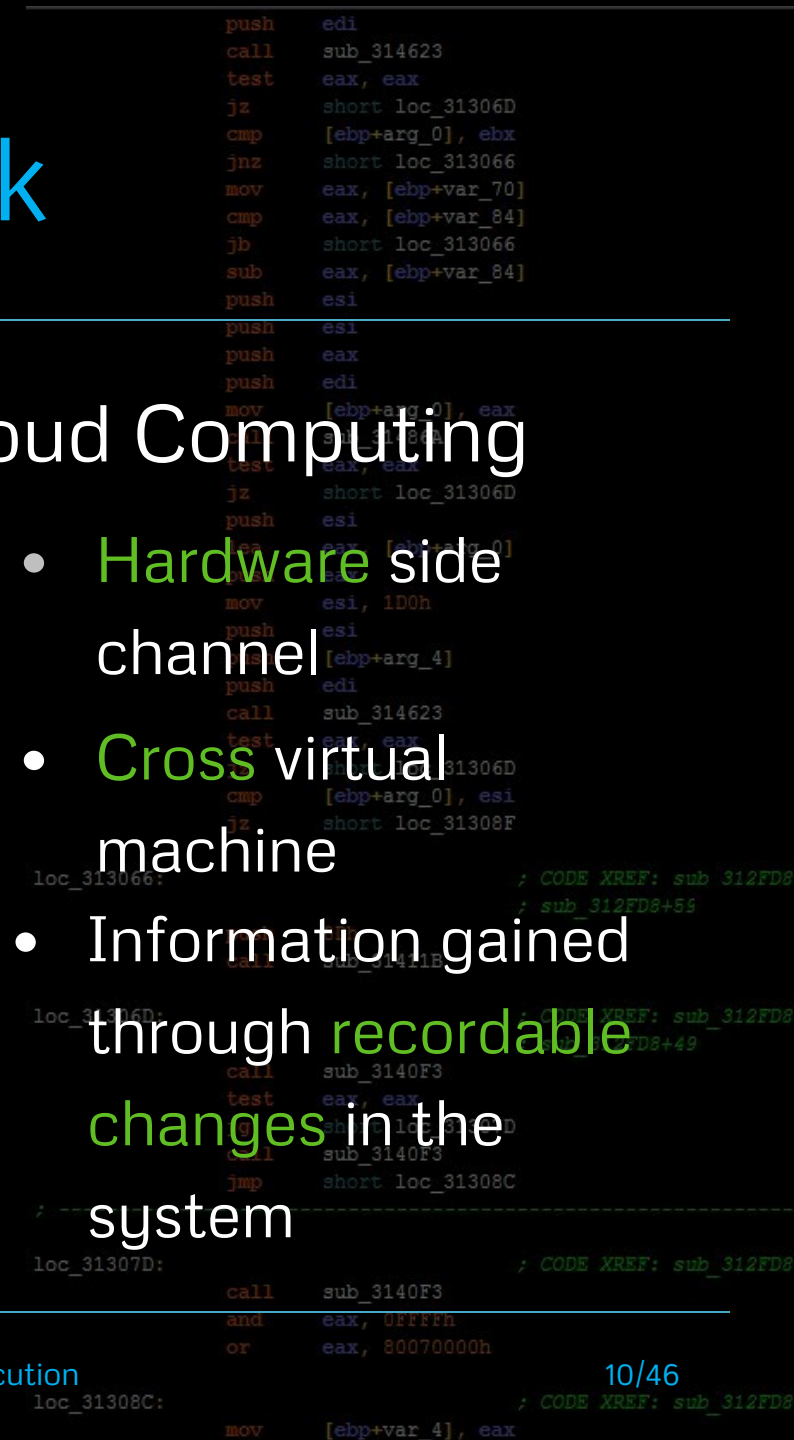


Side Channel Attack

"In cryptography, a **side-channel** attack is any attack based on information gained from the physical implementation of a cryptosystem"

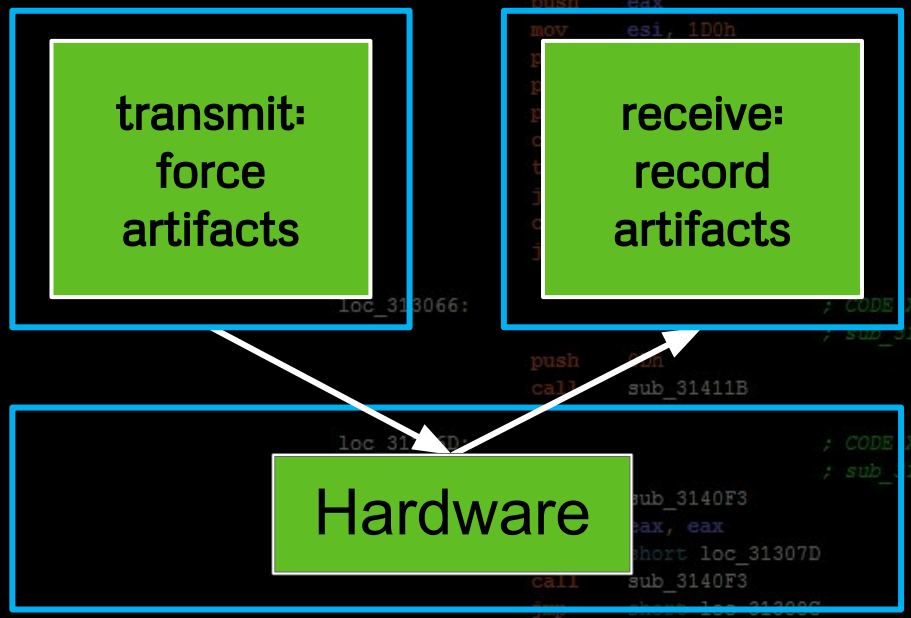
Cloud Computing

- Hardware side channel
- Cross virtual machine
- Information gained through recordable changes in the system



Classification S/R Model

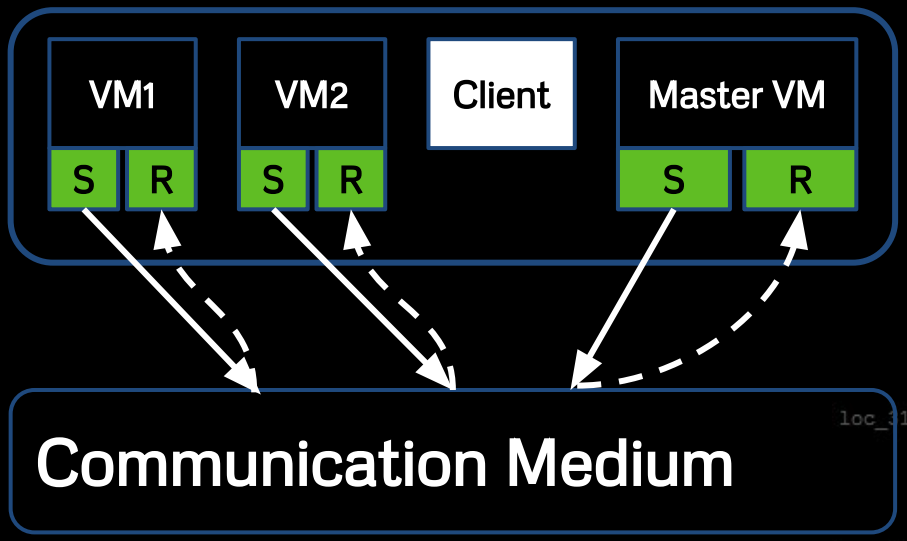
- Hardware agnostic
- Two methods of interacting
  - Transmit
  - Receive



Possible Exploits

- Receive (exfiltrate)
  1. crypto key theft
  2. process monitoring
  3. environment keying
  4. broadcast signal
- Transmit (infiltrate)
  1. DoS
  2. co-residency
- Transmit & Receive (network)
  1. communication (C&C)

Communication



Virtual Allocations

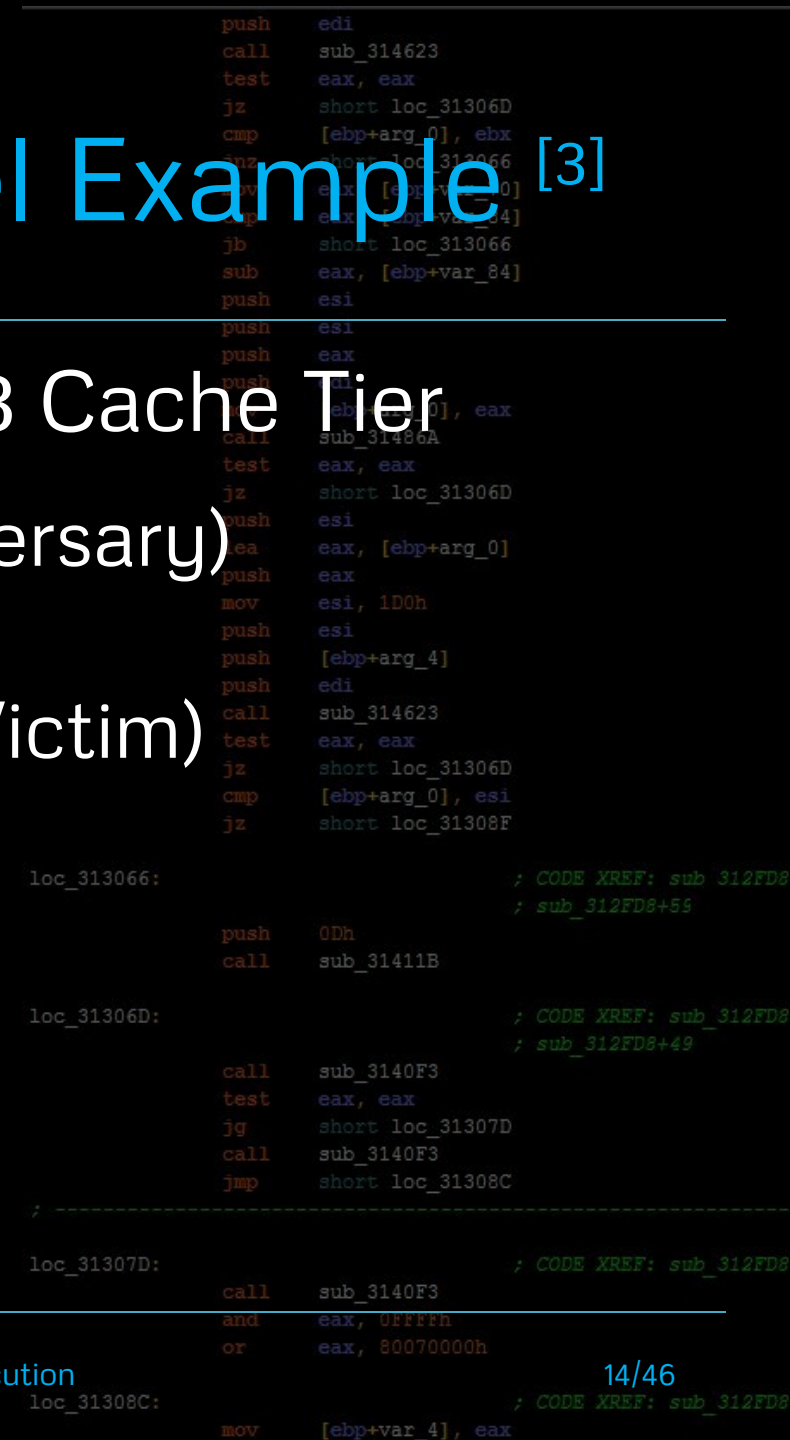
Shared Hardware

Cache Side Channel Example [3]

Flush+Reload targets the L3 Cache Tier

- Receiving Mechanism (Adversary)
  - Flushes & queries
- Transmitting Mechanism (Victim)
  - Accesses same L3 line
- Leaked GnuPG Private Key

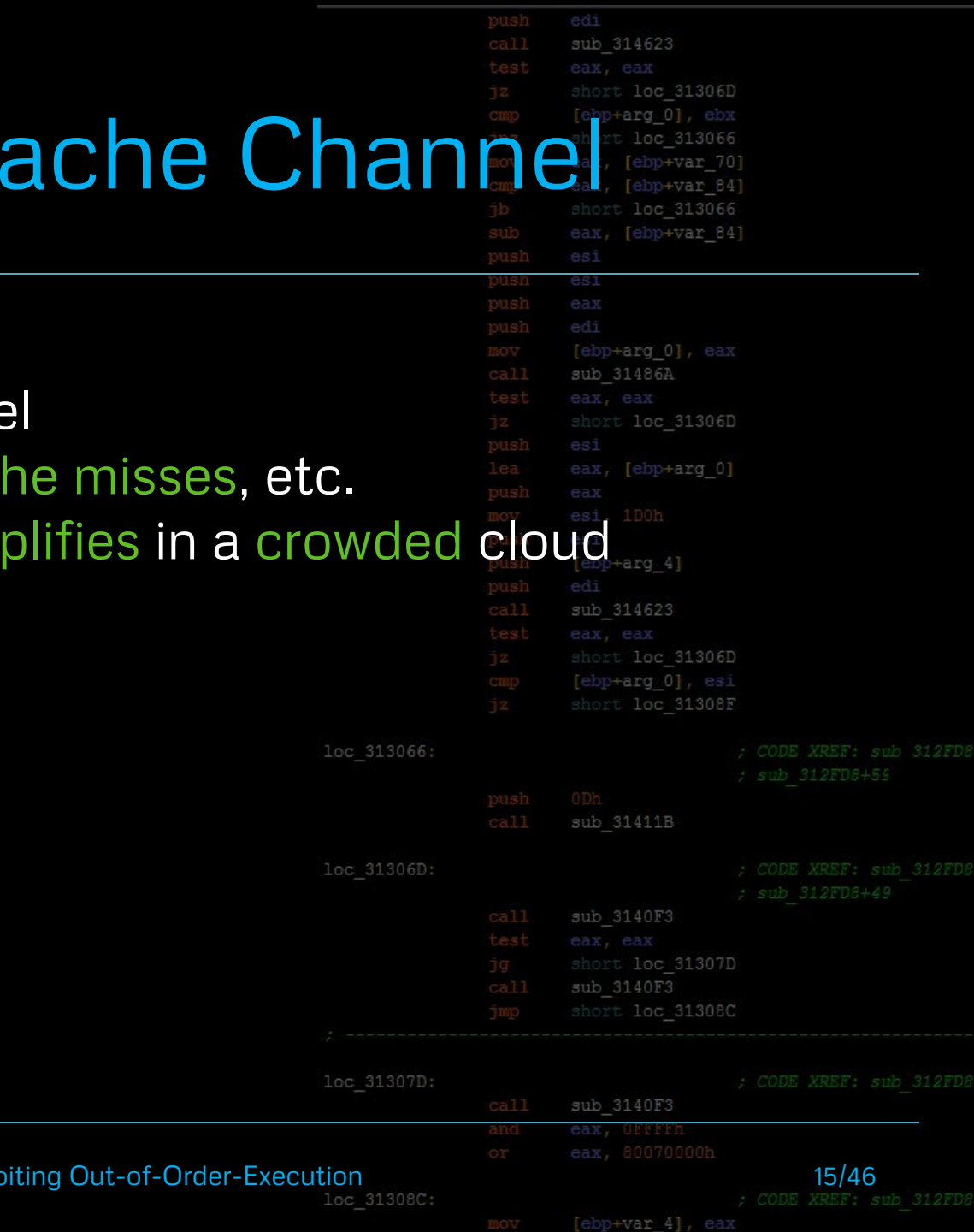
sophia.re/cache.pdf



Pipeline vs Cache Channel

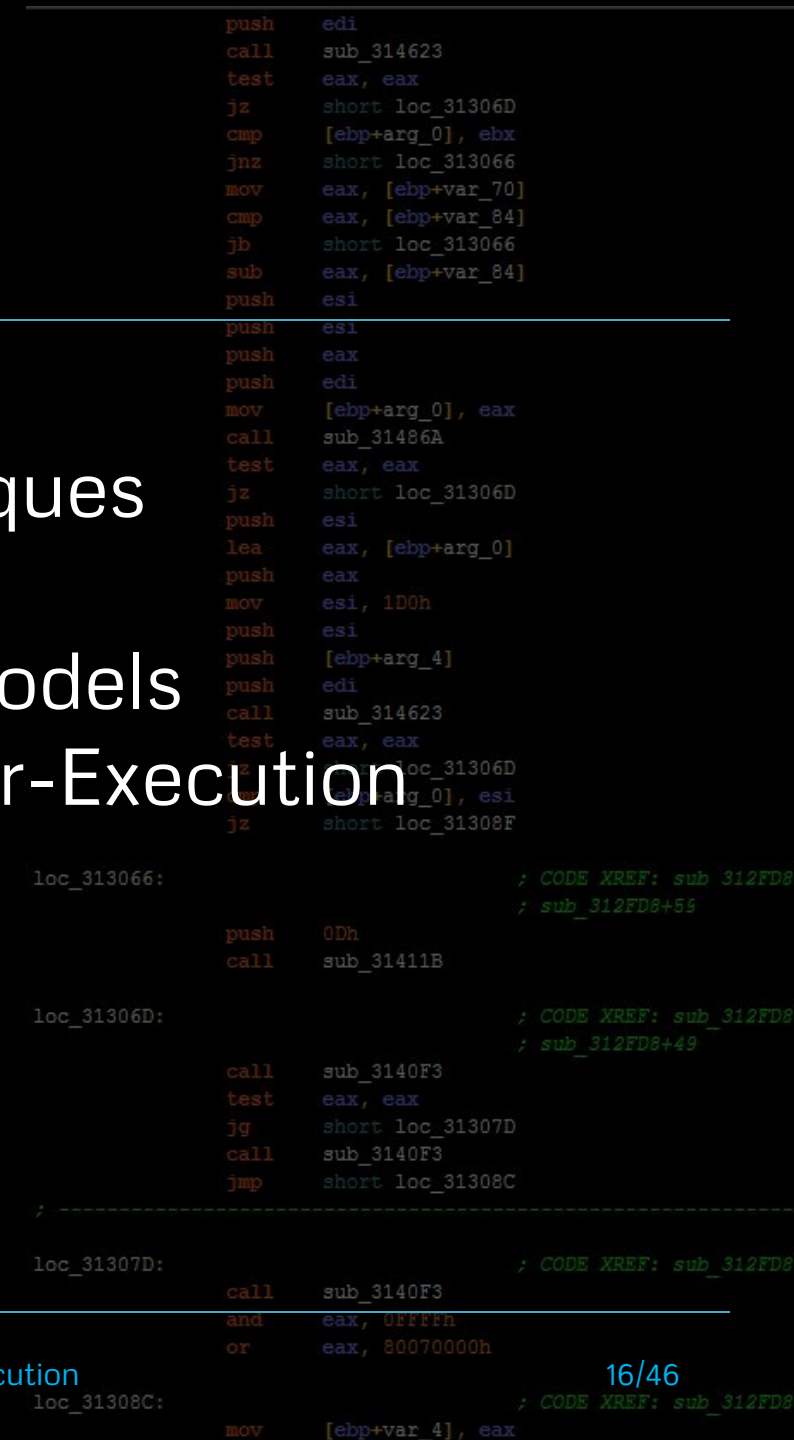
Benefits:

- Quiet, covert channel
- Not affected by cache misses, etc.
- Channel & noise amplifies in a crowded cloud environment



Overview

1. Introduction
2. Cloud exploitation techniques
3. Targeting the pipeline
4. Importance of memory models
5. Design of an Out-of-Order-Execution channel
6. Demo
7. Conclusion



The Attack Vector

Side Channels which Exploit Hardware Vulnerabilities Inherent to Modern Cloud Computing Systems

Requirements:

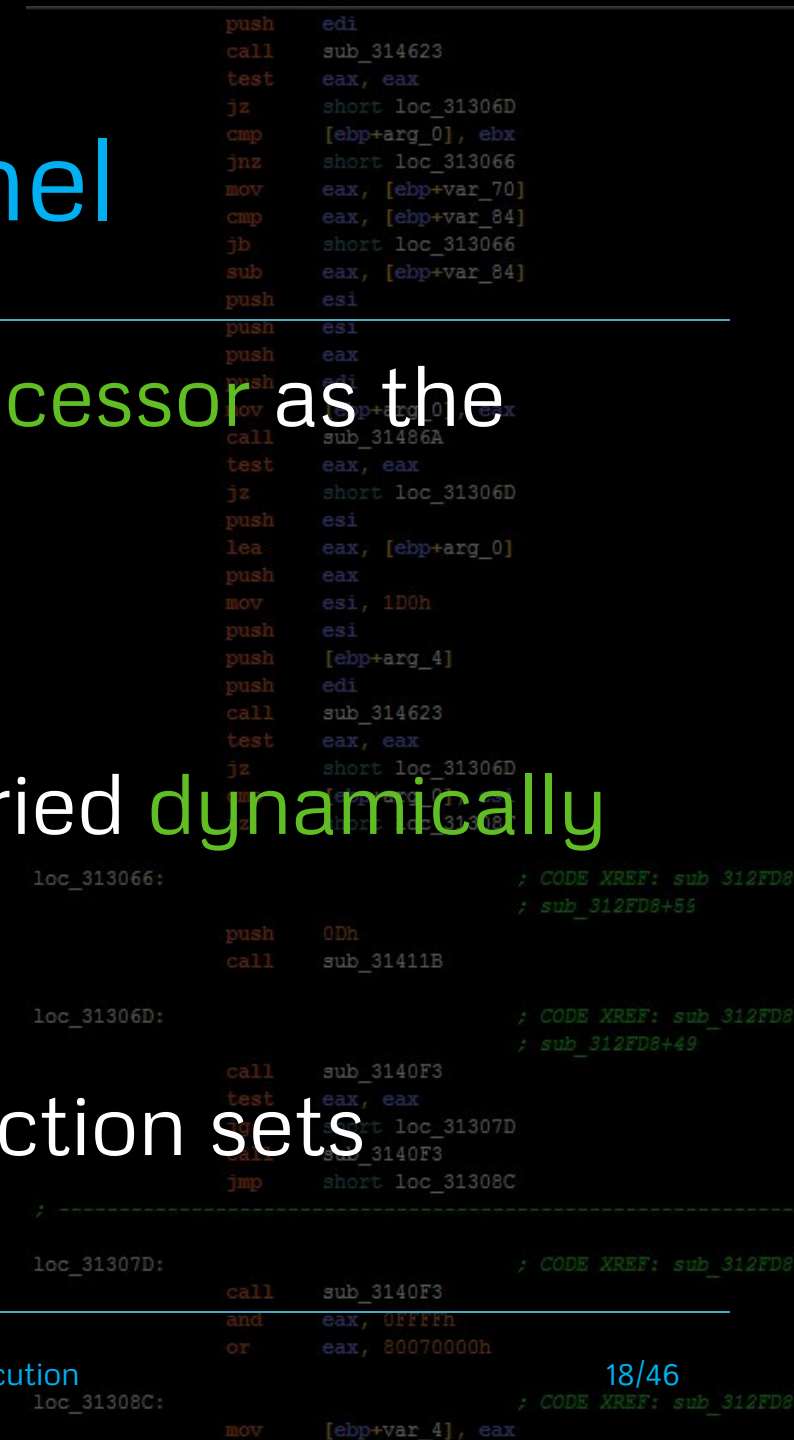
- Shared hardware
- Dynamically allocated hardware resources
- Co-Location with adversarial VMs or infected VMs

Pipeline Side Channel

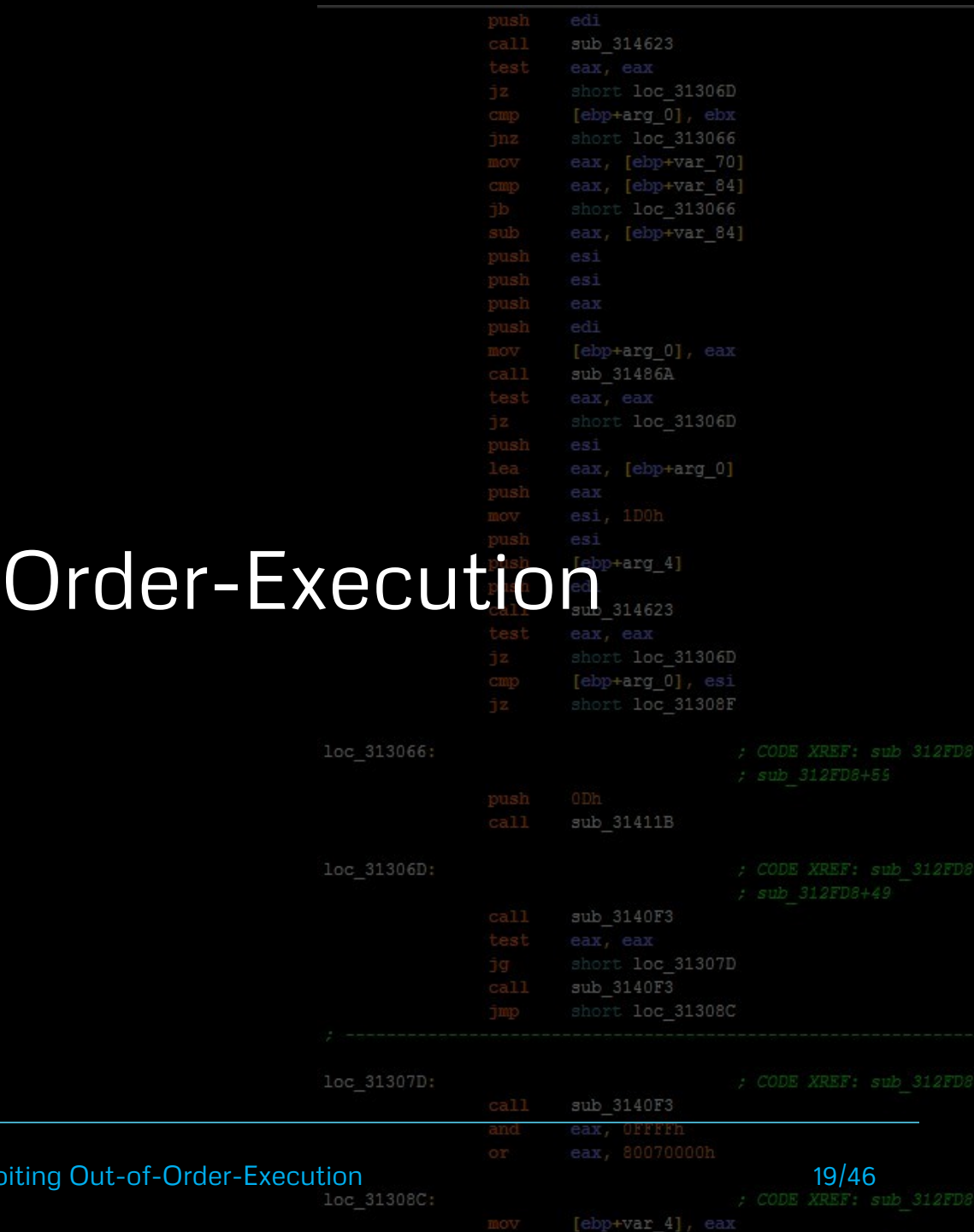
We chose to target the processor as the hardware medium.

=> CPU's pipeline
=> System artifacts queried dynamically

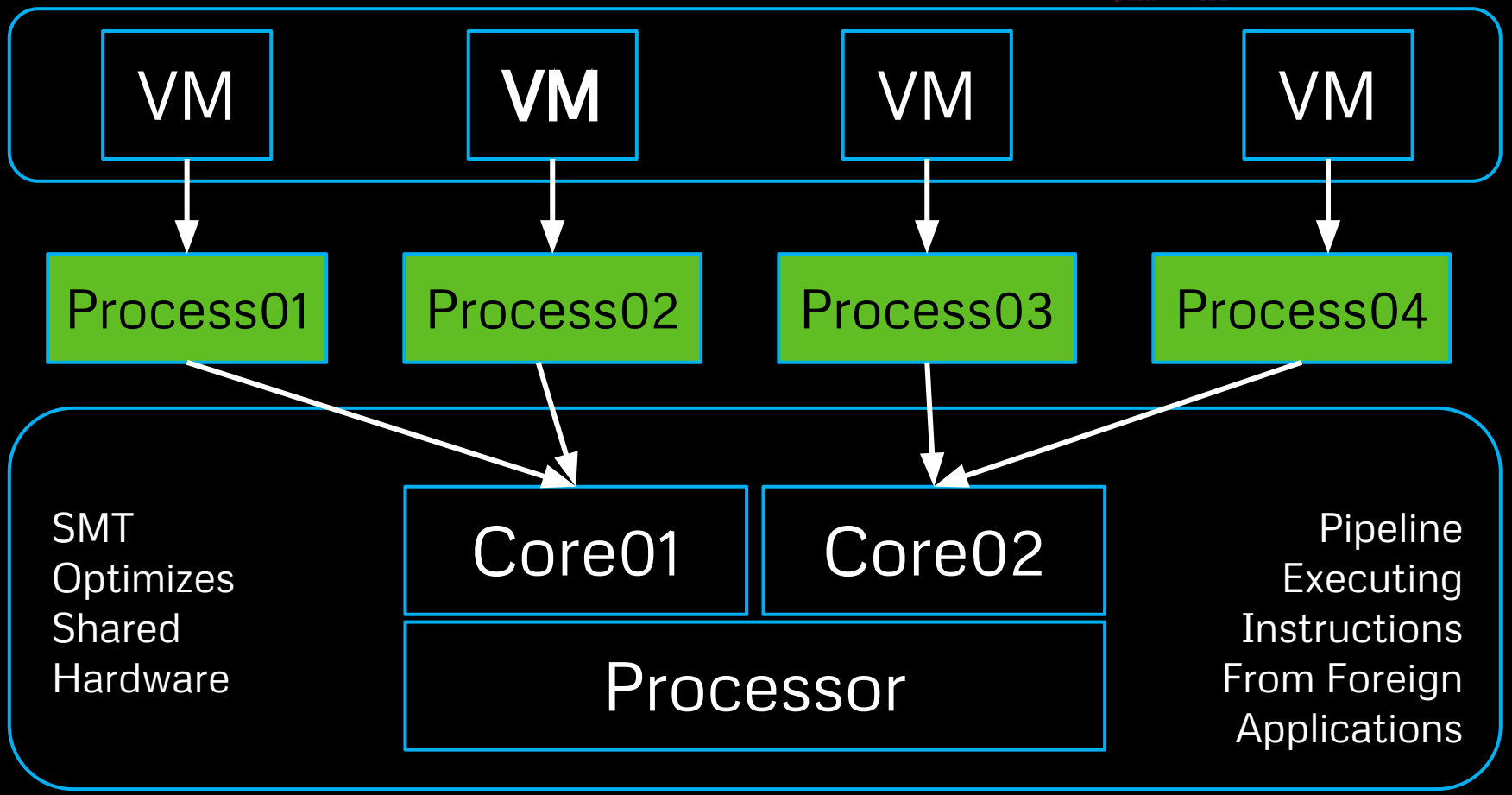
- Instruction order
- Results from instruction sets



Out-of-Order-Execution



Processor Pipeline Contention



RECEIVER

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h
loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Record Out of Order Execution [6]

8.2.3.4 Loads May Be Reordered with Earlier Stores to Different Locations

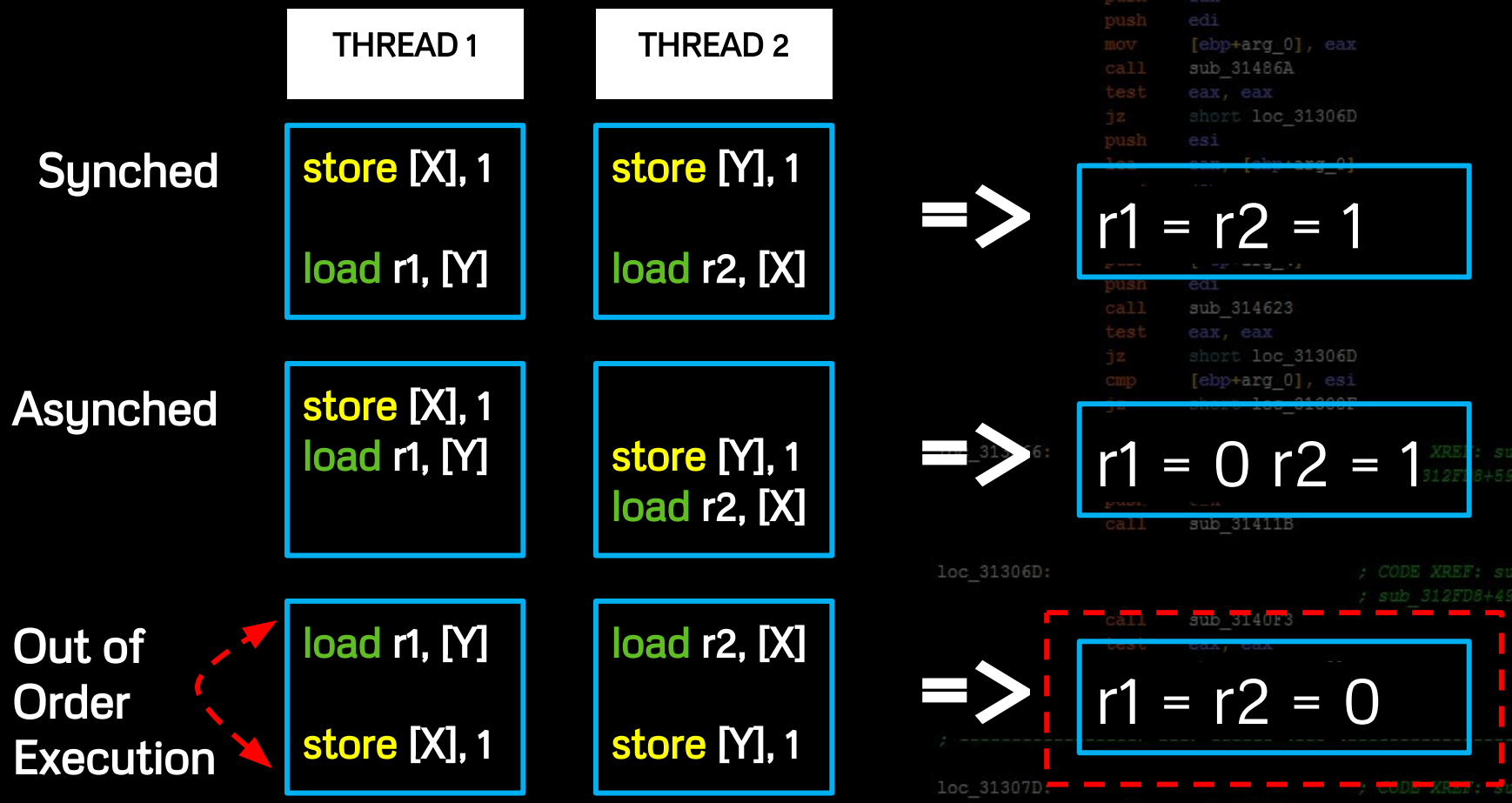
The Intel-64 memory-ordering model allows a load to be reordered with an earlier store to a different location. However, loads are not reordered with stores to the same location.

The fact that a load may be reordered with an earlier store to a different location is illustrated by the following example:

Example 8-3. Loads May be Reordered with Older Stores

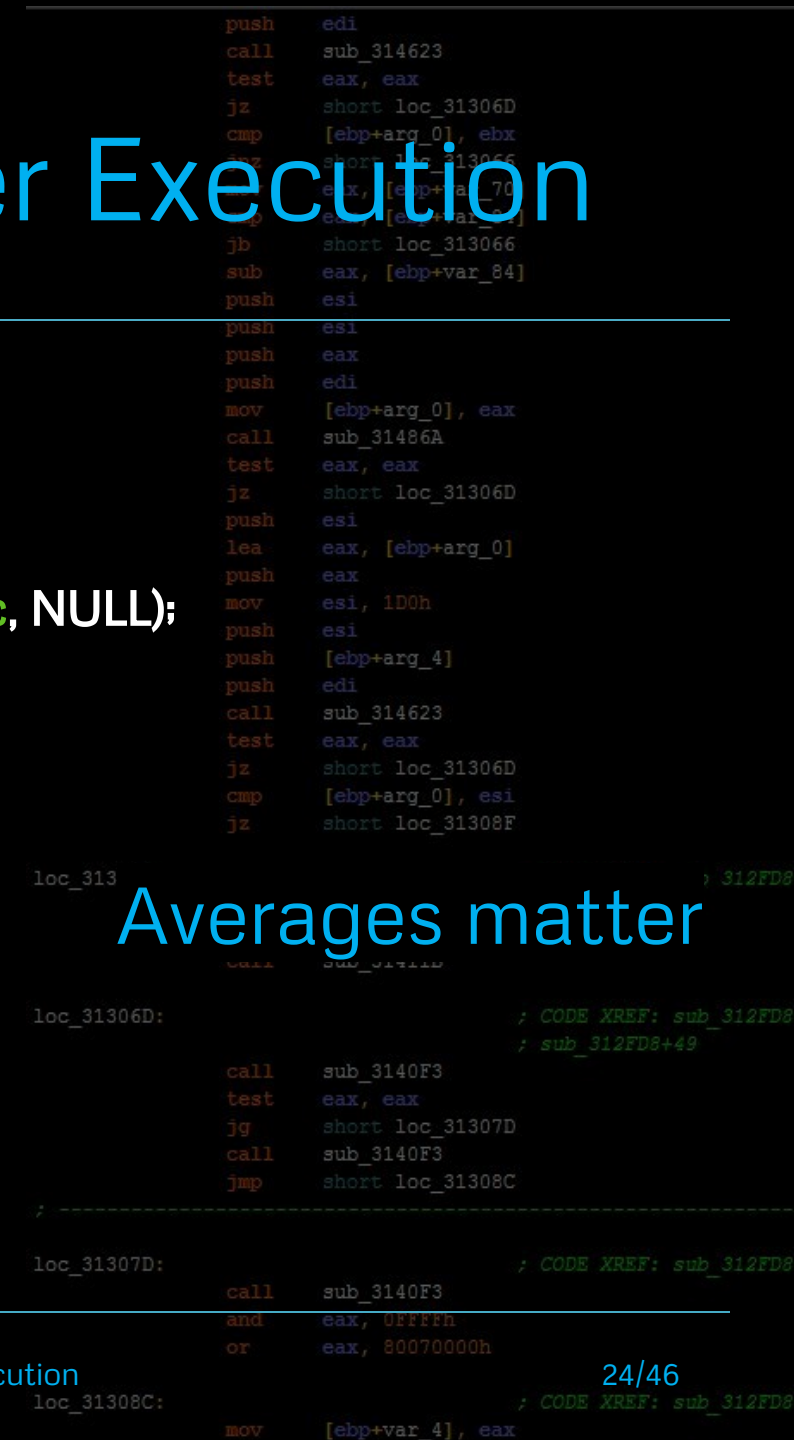
Processor 0	Processor 1
mov [_x], 1 mov r1, [_y]	mov [_y], 1 mov r2, [_x]
Initially x = y = 0 r1 = 0 and r2 = 0 is allowed	

Record Out of Order Execution



Record Out of Order Execution

```
int X,Y,count_OoOE;
....initialize semaphores Sema1 & Sema2...
pthread_t thread1, thread2;
pthread_create(&threadN, NULL, threadNFunc, NULL);

for (int iterations = 1; ; iterations++)
        X,Y = 0;
        sem_post(beginSema1 & beginSema2);
        sem_wait(endSema1 & endSema2);

         if (r1 == 0 && r2 == 0)
              count_OoOE ++;
```



Averages matter

TRANSMITTER

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h
loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Force Out of Order Execution

Memory Fences

Mfence:

- x86 instruction full memory barrier
  prevents memory reordering of any kind
- order of 100 cycles per operation

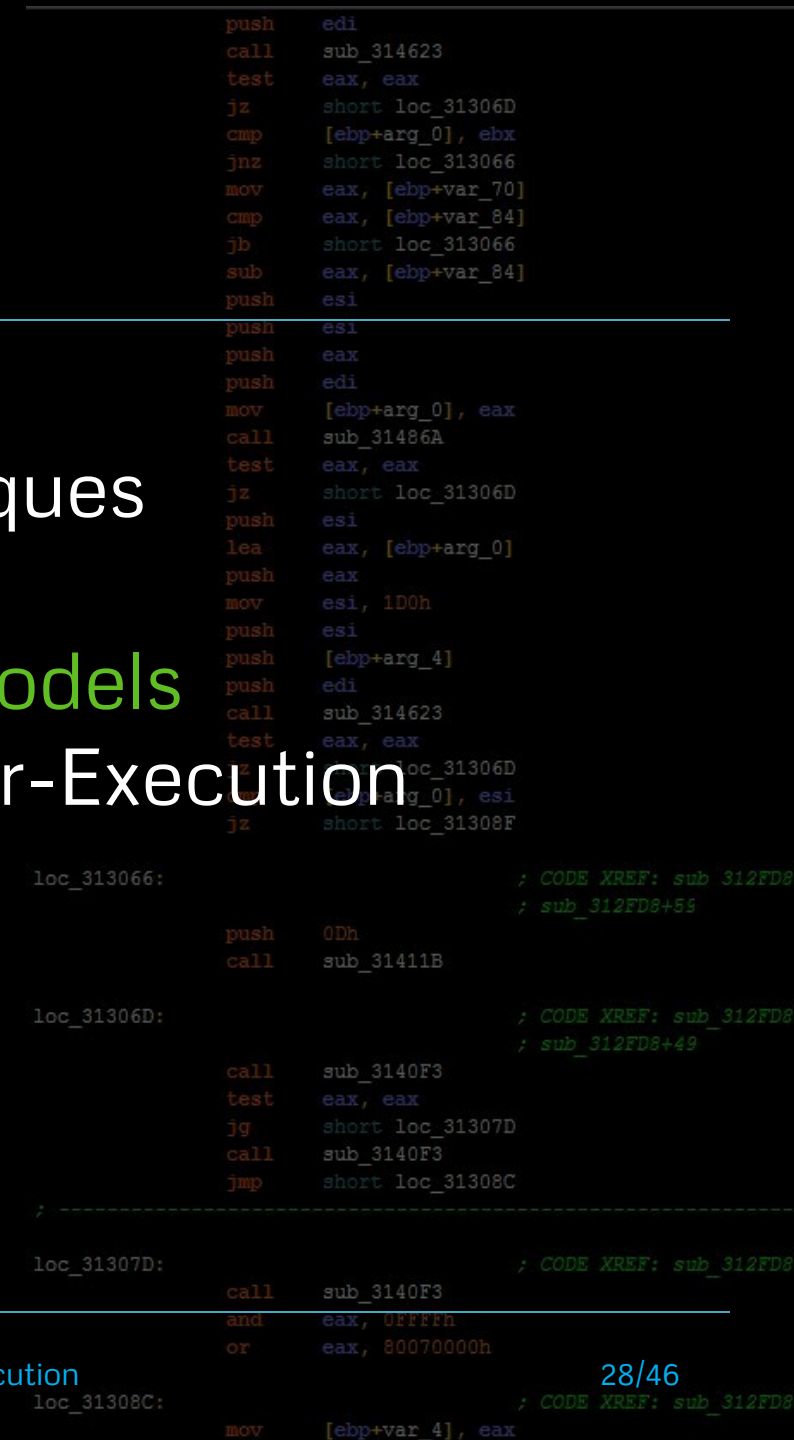
```
... mov dword ptr [_spin1], 0
    ...   mfence

... mov dword ptr [_spin2], 0
    ...   mfence
```

Force Out of Order Execution

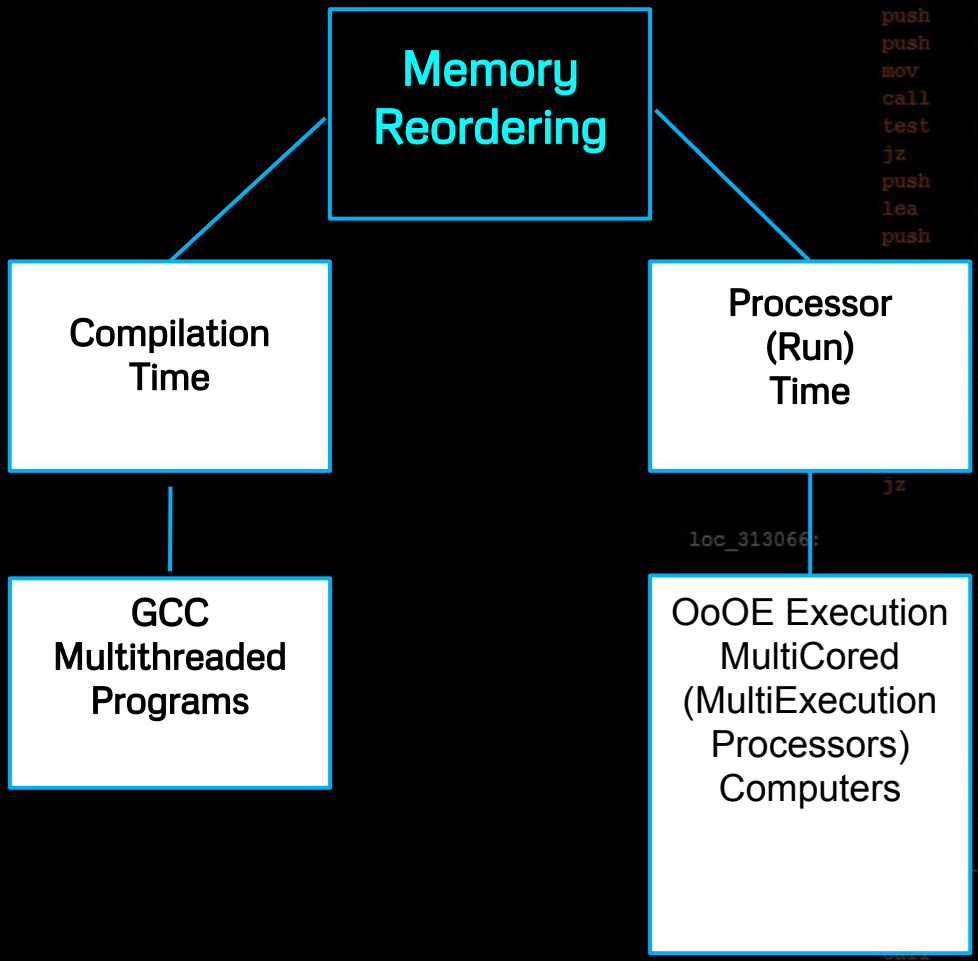
THE PIPELINE

Overview

1. Introduction
2. Cloud exploitation techniques
3. Targeting the processor
4. Importance of memory models
5. Design of an Out-of-Order-Execution channel
6. Demo
7. Conclusion

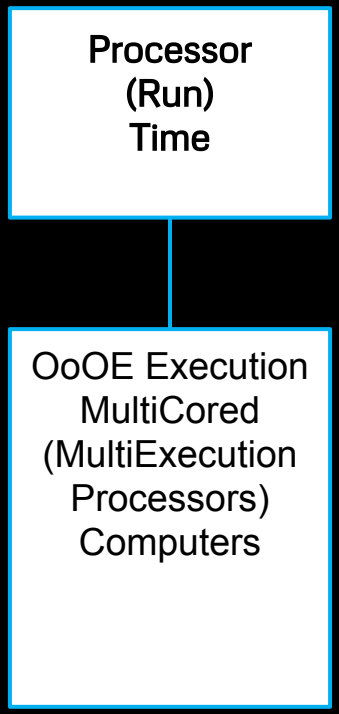


Types of Memory Reordering

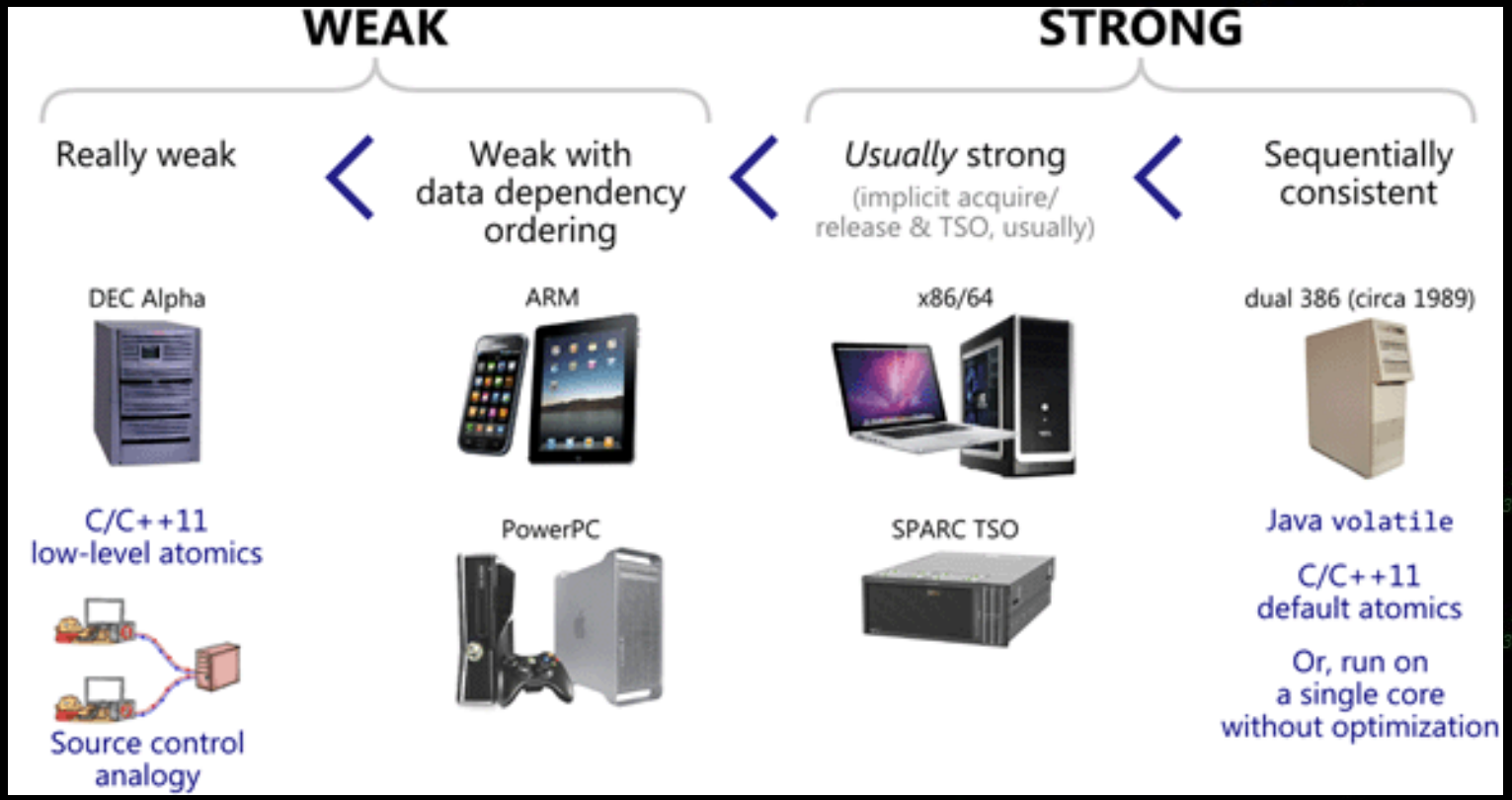


Types of Memory Reordering

Dynamic side channel artifacts

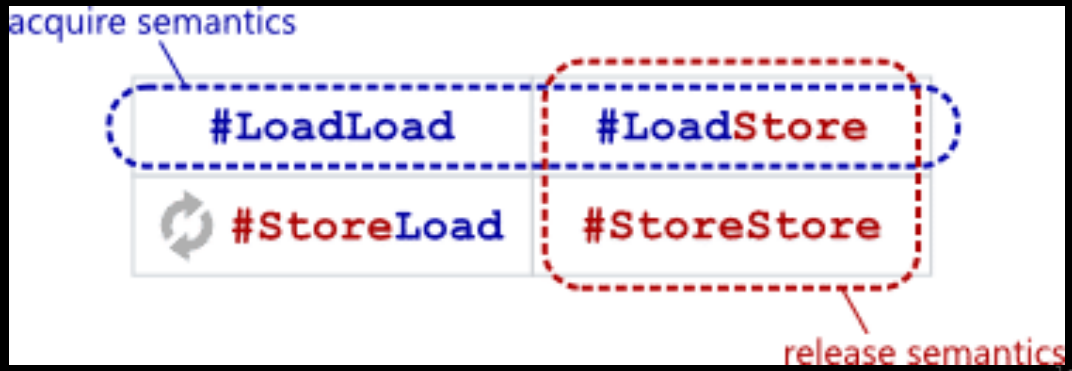


Weak Memory Models [7]



Types of Memory Reordering

4 types of run time reordering barriers



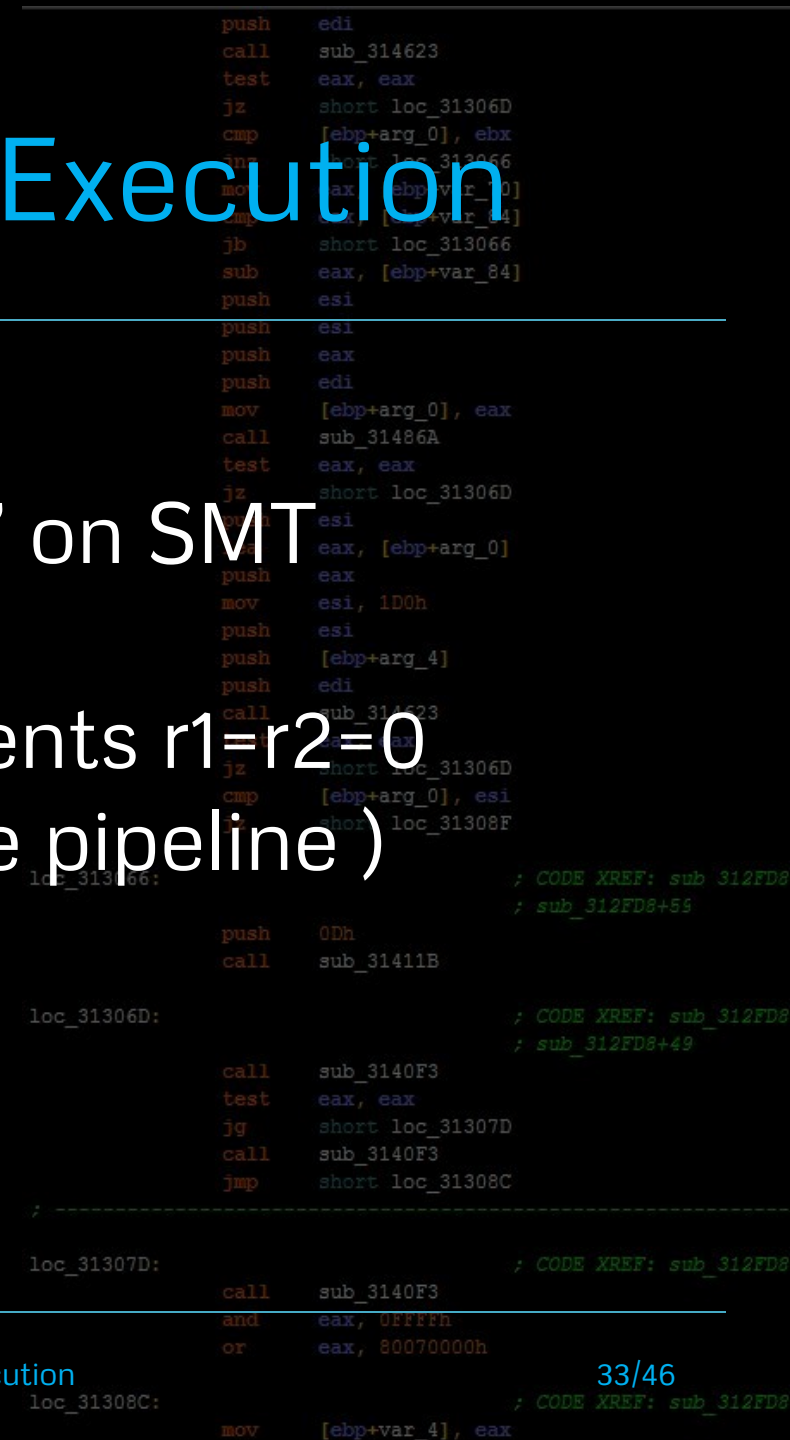
[4, 5]

- Instruction A visible to all processes before B occurs
- #StoreLoad most expensive operation

Force Out of Order Execution

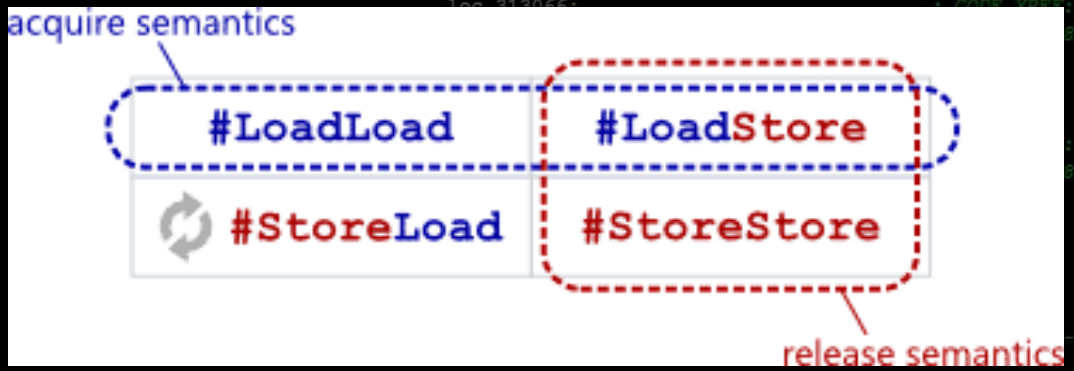
Memory Barrier

- ‘Lock-free programming’ on SMT multiprocessors
- #StoreLoad unique prevents r1=r2=0
- x86: mfence (effects the pipeline)



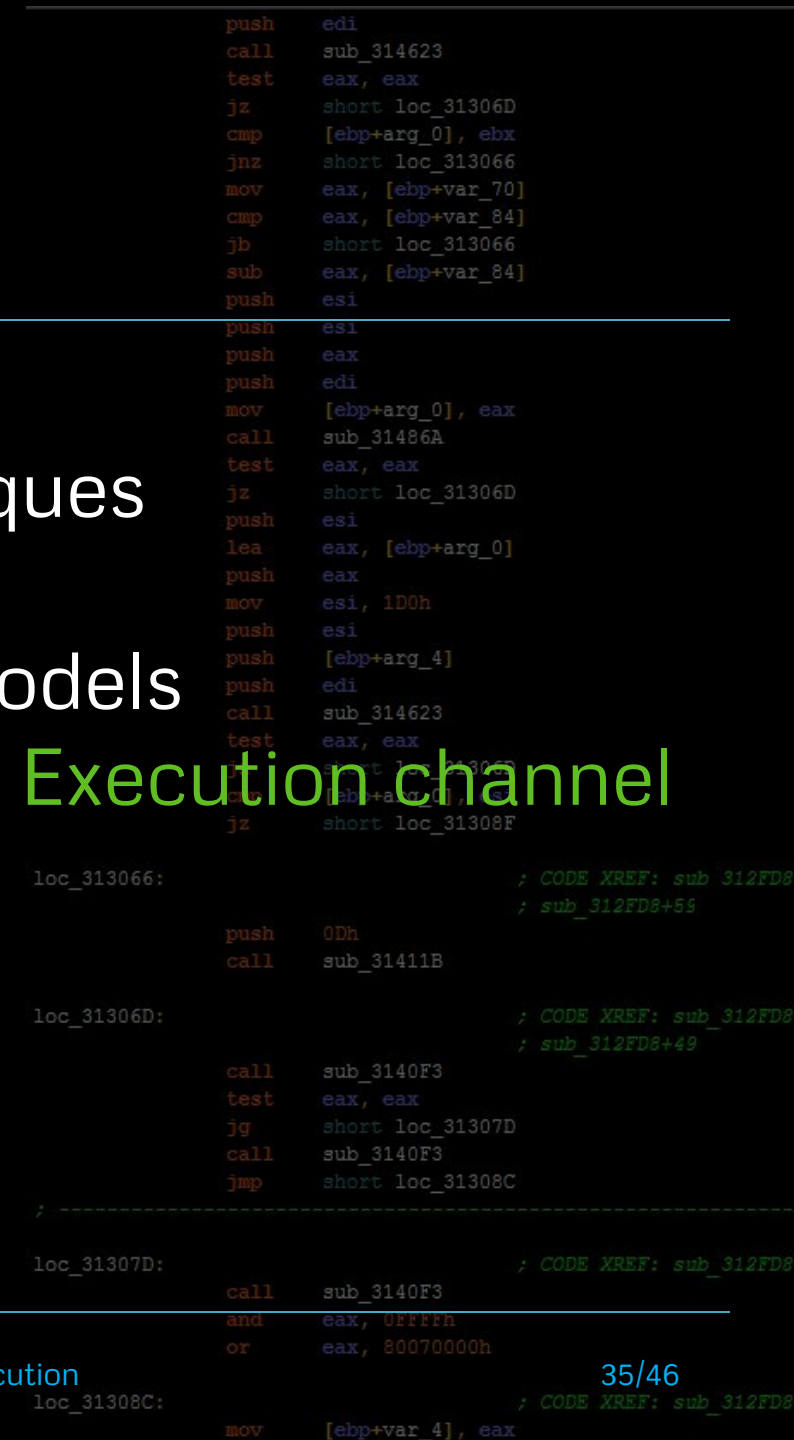
Channel Transmitter (Victim)

- Force Out-of-Order-Execution patterns
- Affect the order of stores and loads
- Time frame dependant
- x86: mfence



Overview

1. Introduction
2. Cloud exploitation techniques
3. Targeting the processor
4. Importance of memory models
5. Design of an Out of Order Execution channel
6. Demo
7. Conclusion



Lab Model

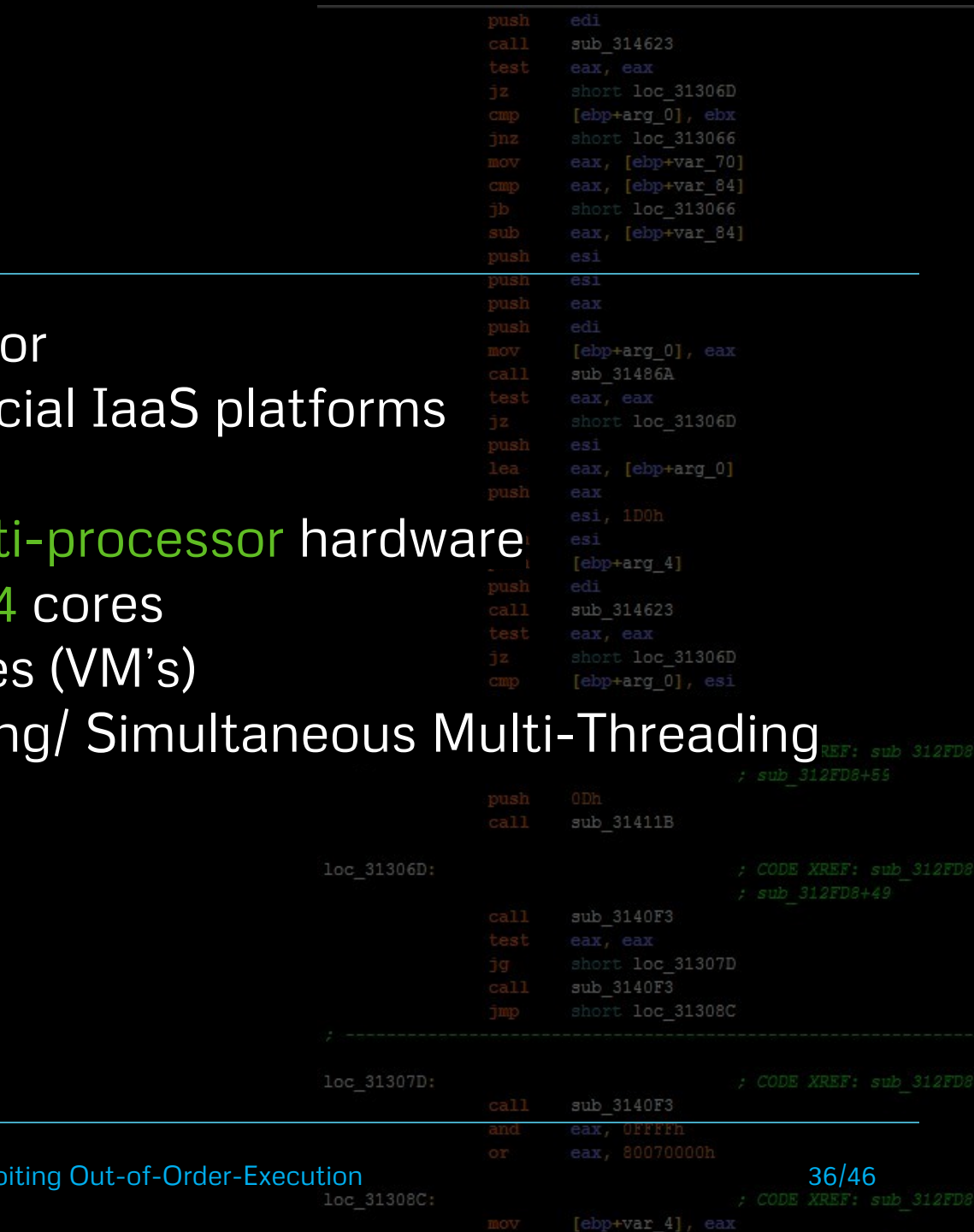
Scheduler Xen hypervisor

- Popular commercial IaaS platforms

Xeon Processors

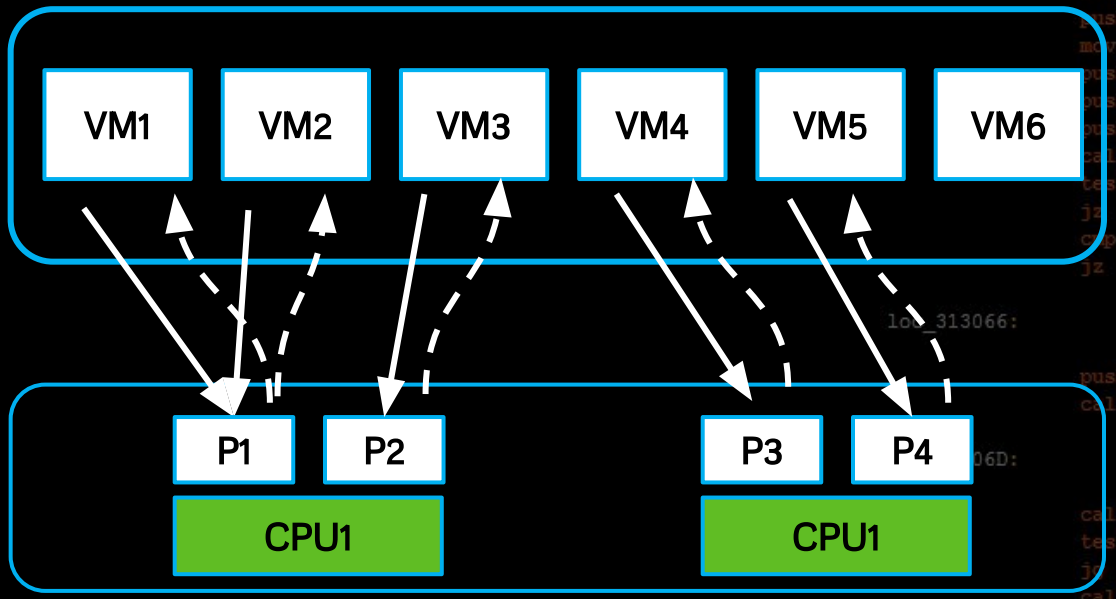
Shared multi-core/ multi-processor hardware

- 8 logical CPU's/ 4 cores
- 6 virtual machines (VM's)
- Parallel Processing/ Simultaneous Multi-Threading On (SMT)

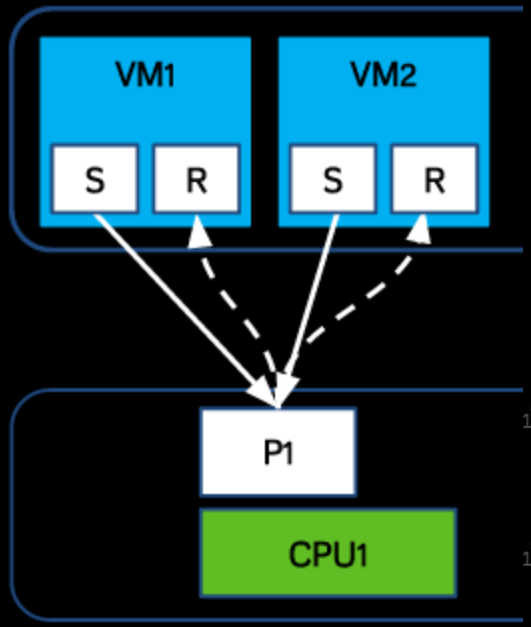


Virtual Machines

- 6 Windows 7 VM's



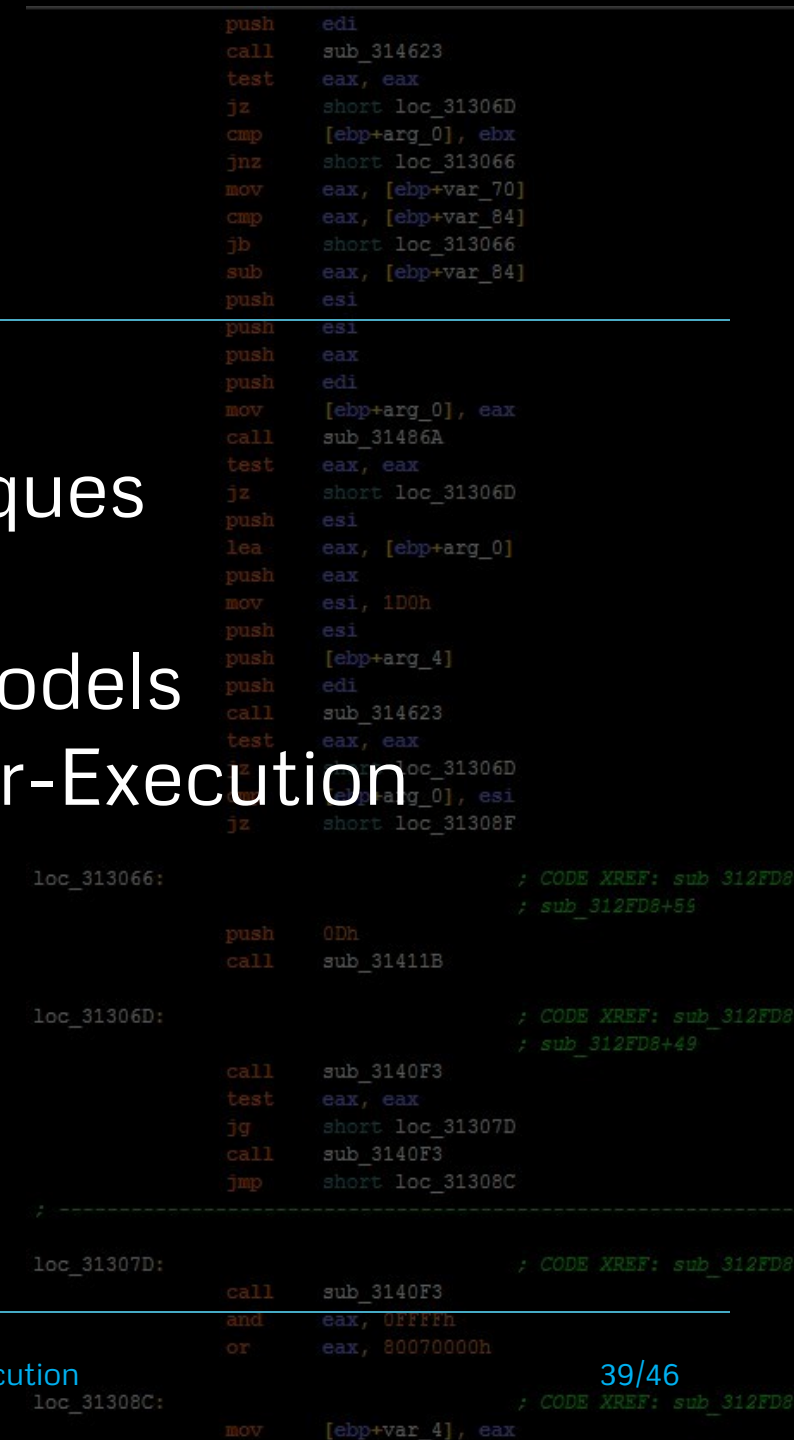
Virtual Machine S/R




```
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], ebx
jnz     short loc_313066
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jb      short loc_313066
sub     eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
mov     [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz      short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], esi
jz      short loc_31308F

13066:                                  ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
push    0Dh
call    sub_31411B

1306D:                                  ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg      short loc_31307D
call    sub_3140F3
jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
call    sub_3140F3
and     eax, 0FFFFh
or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
mov     [ebp+var_4], eax
```

Overview

1. Introduction
2. Cloud exploitation techniques
3. Targeting the processor
4. Importance of memory models
5. Design of an Out-of-Order-Execution channel
6. Demo
7. Conclusion



Demo Links

sophia.re/sender.py

sophia.re/receiver.py

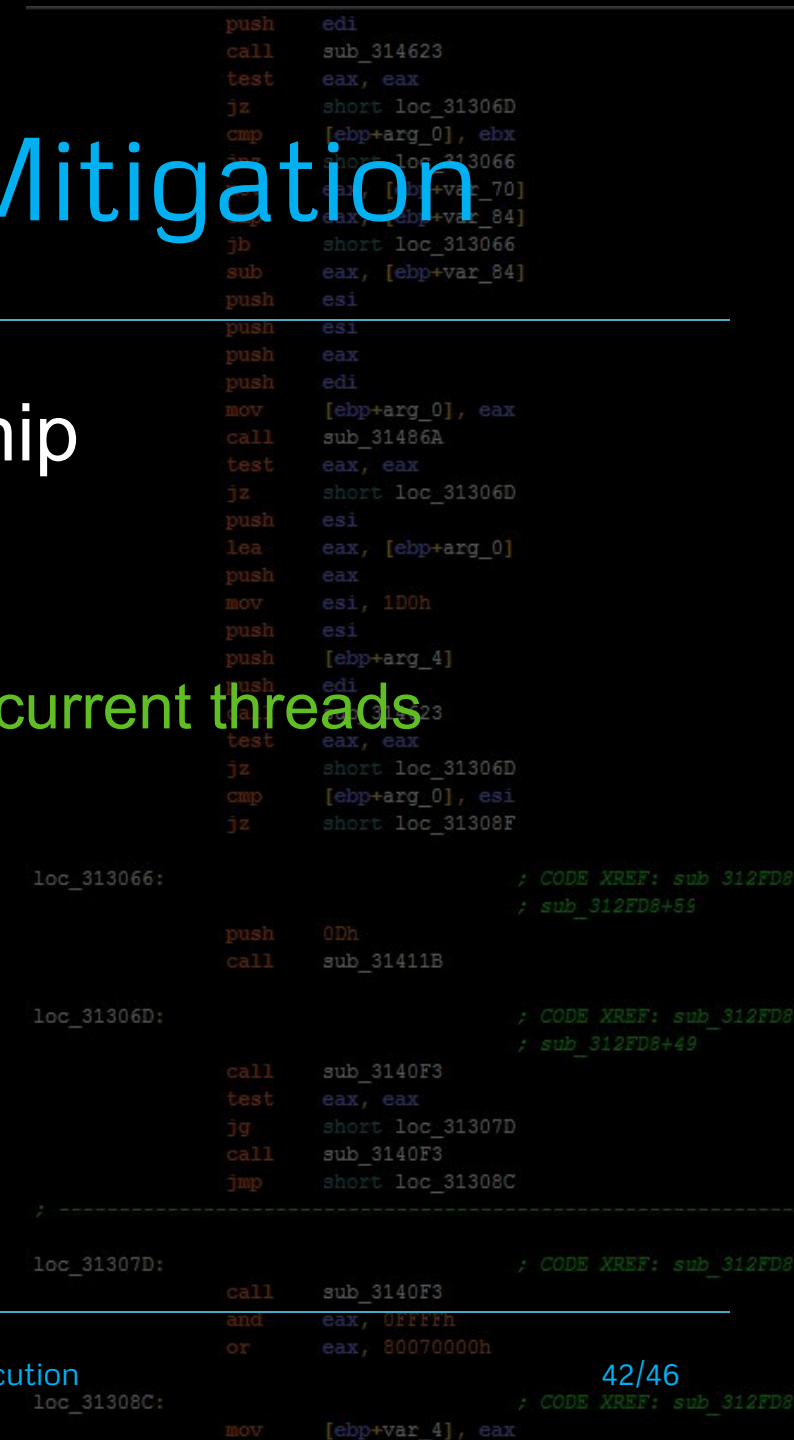
Overview

1. Introduction
2. Cloud exploitation techniques
3. Targeting the processor
4. Importance of memory models
5. Design of an Out-of-Order-Execution channel
6. Demo
7. Conclusion

Potential Channel Mitigation

Protected Resource Ownership

- Isolating VM's
- Turn off hyperthreading
- Blacklisting resources for concurrent threads
- Downside: cloud benefits

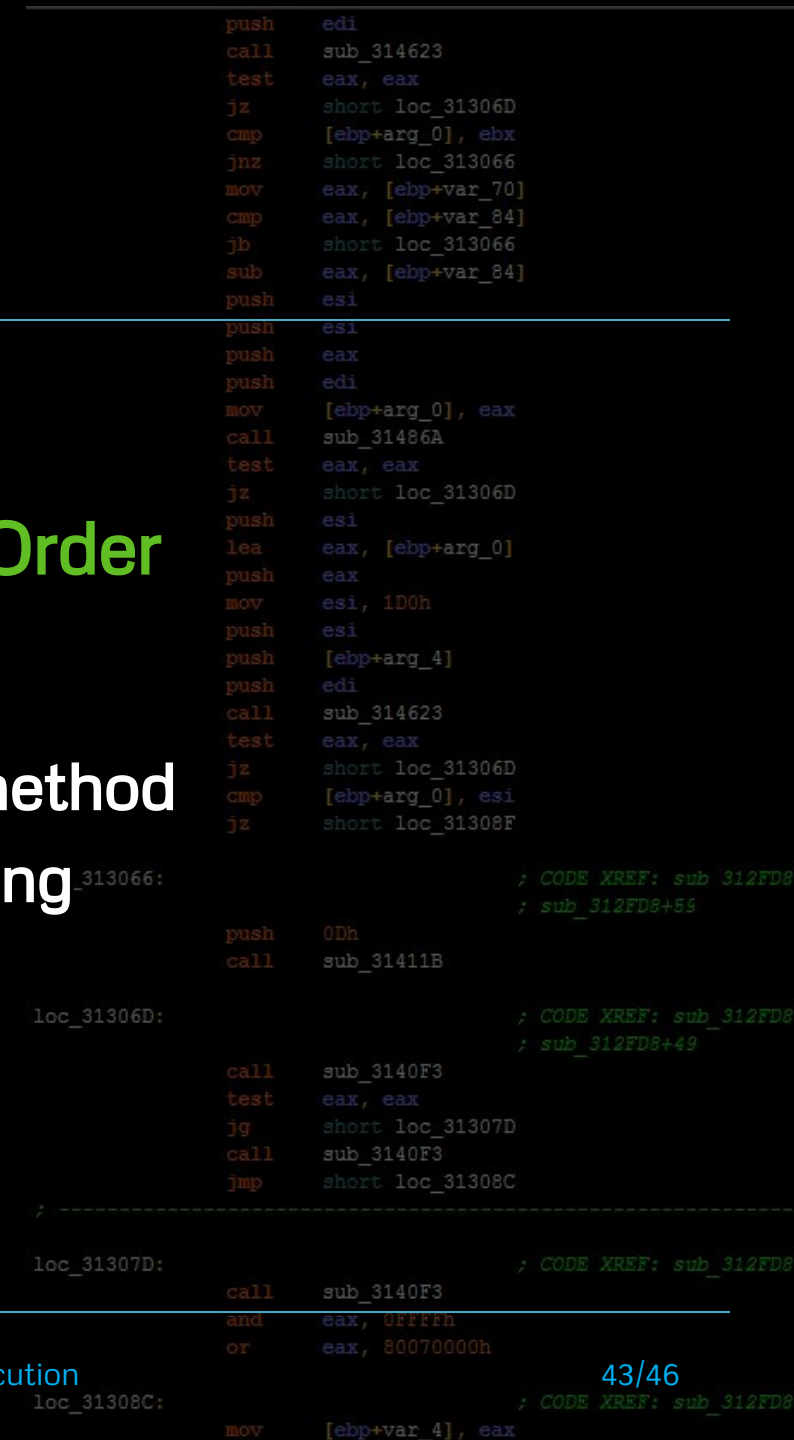


In Conclusion…

Contribution:

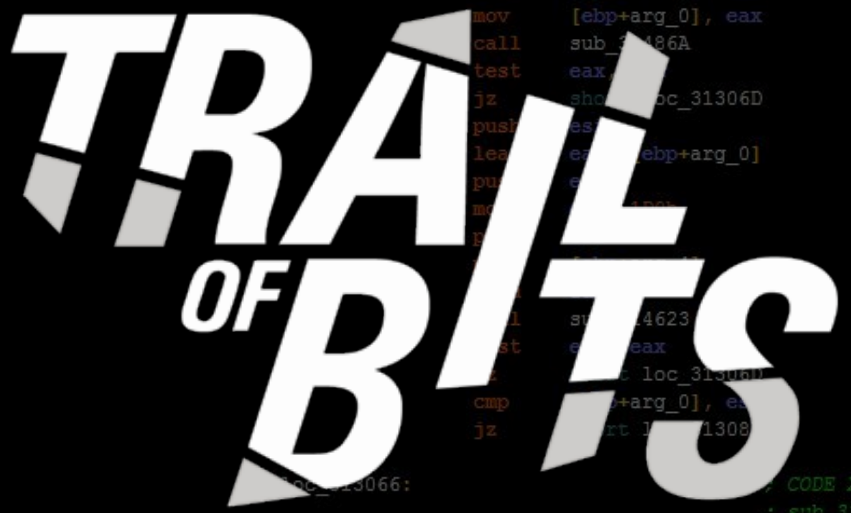
We demonstrate a novel **Out of Order Execution** side channel.

- **Dynamic** querying/ forcing method
- **Application** to cloud computing
- **Mitigation** techniques

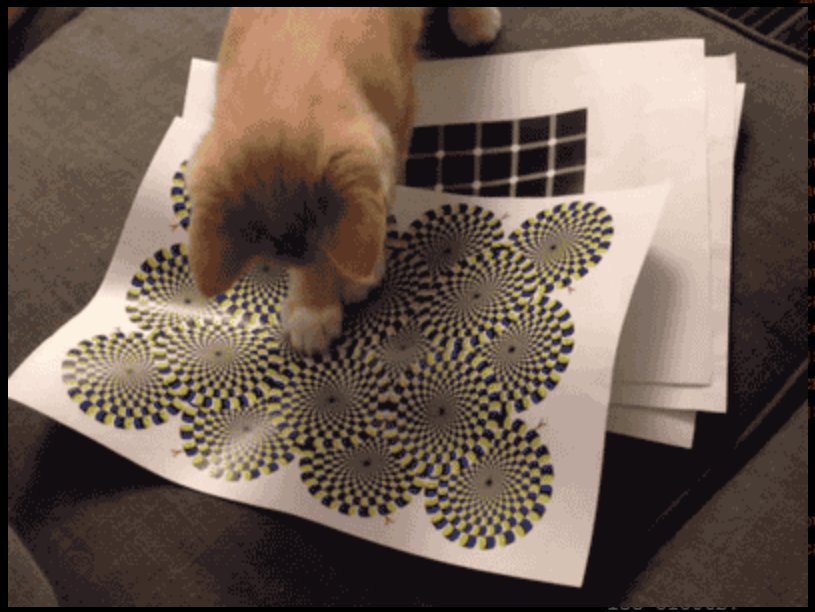


Acknowledgements

- Jeremy Blackthorne
- RPISEC
- Trail of Bits



Any Questions?



IRC: quend (#rpisec, #pwning)
email: sophia@trailofbits.com
thesis link: sophia.re/thesis.pdf

References

[1] http://www.thewhir.com/web-hosting-news/aws-to-reach-24-billion-in-revenue-by-2022-morgan-stanley

[2] http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/

[3] https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-yarom.pdf

[4] http://bartoszmilewski.com/2008/11/05/who-ordered-memory-fences-on-an-x86/

[5] http://preshing.com/20120913/acquire-and-release-semantics/

[6] http://www.intel.com/Assets/en_US/PDF/manual/253668.pdf

[7] http://preshing.com/20120930/weak-vs-strong-memory-models/

[8] http://en.wikipedia.org/wiki/Memory_barrier#An_illustrative_example

[9] http://preshing.com/20120710/memory-barriers-are-like-source-control-operations/