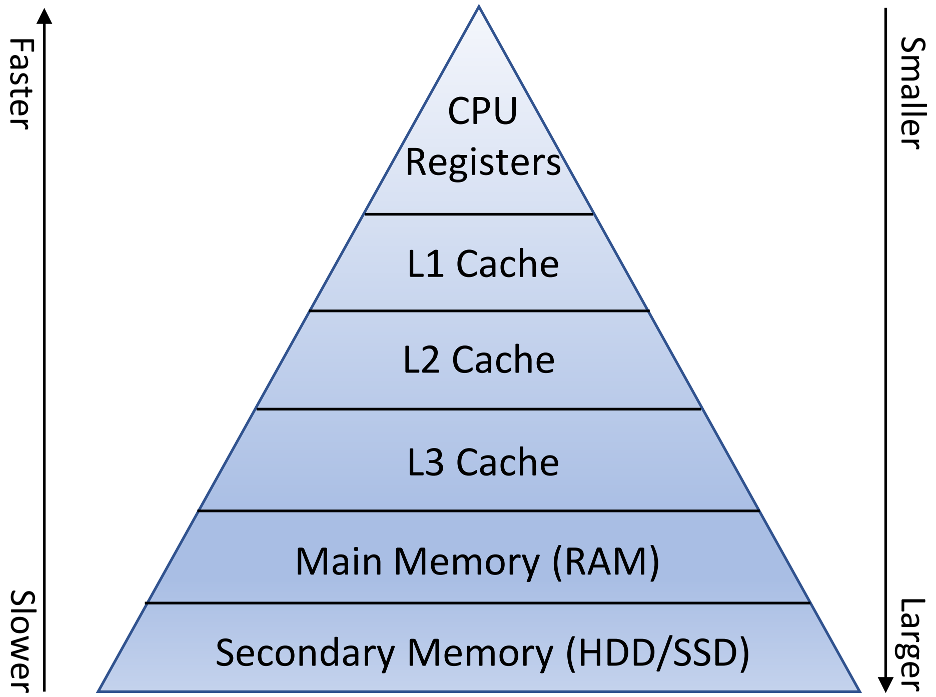
Diminisher: A Linux Kernel-based Countermeasure for TAA Vulnerability

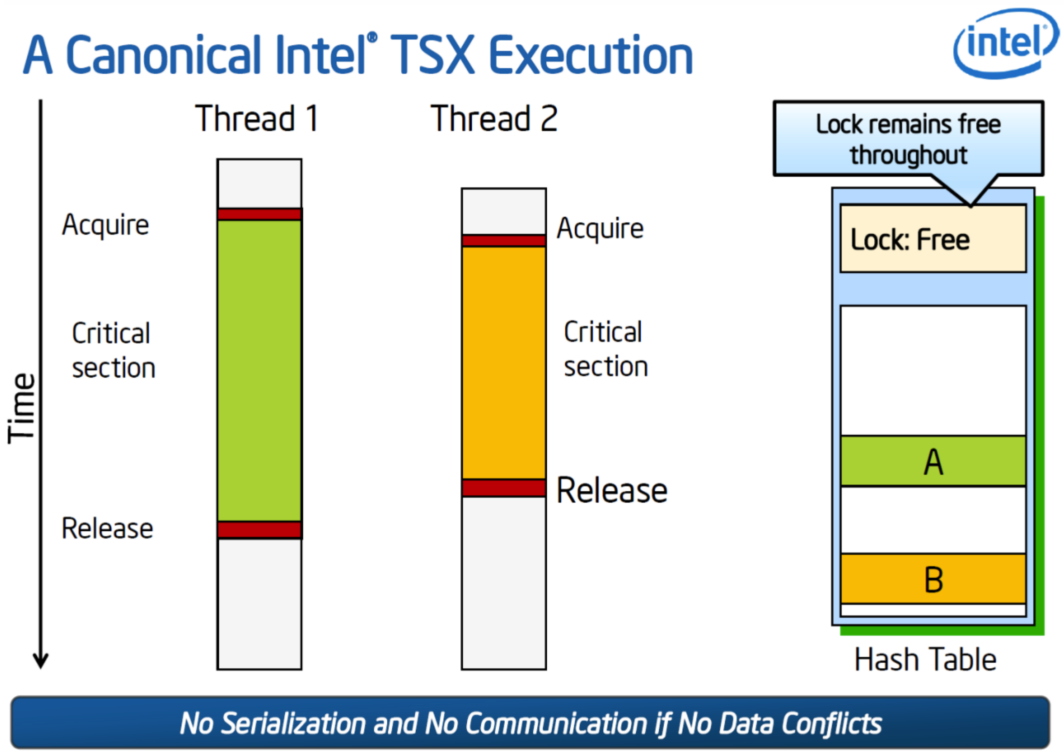
Ameer Hamza (ITU), Maria Mushtaq (Télécom Paris), Khurram Bhatti (ITU), David Novo (LIRMM), Florent Bruguier (LIRMM), Pascal Benoit (LIRMM)

Background

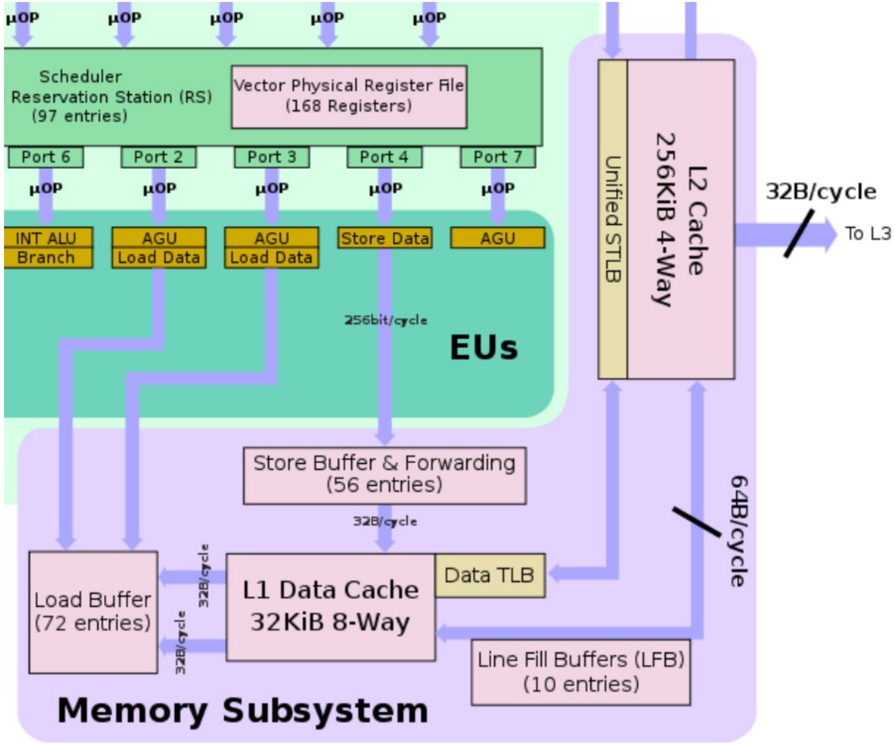
Caches


Faster
Slower
CPU Registers
L1 Cache
L2 Cache
L3 Cache
Main Memory (RAM)
Secondary Memory (HDD/SSD)
Smaller
Larger

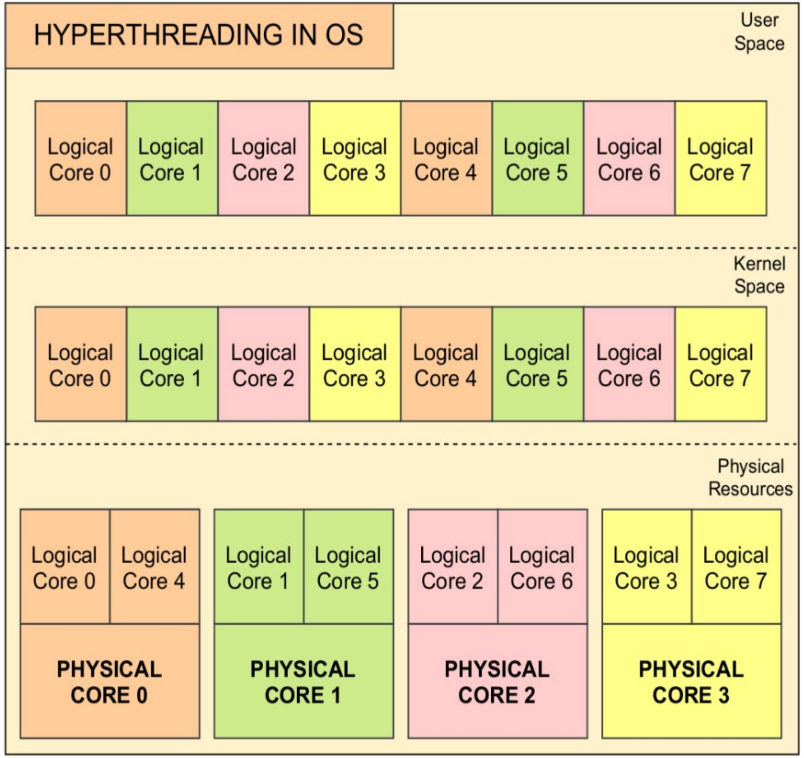
Intel TSX


A Canonical Intel® TSX Execution
intel
Time
Thread 1
Thread 2
Acquire
Critical section
Release
Acquire
Critical section
Release
Lock remains free throughout
Lock: Free
A
B
Hash Table
No Serialization and No Communication if No Data Conflicts

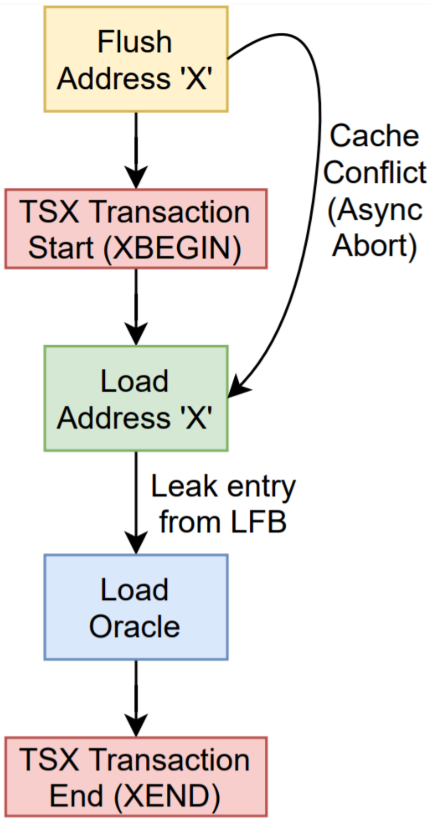
Line Fill Buffer



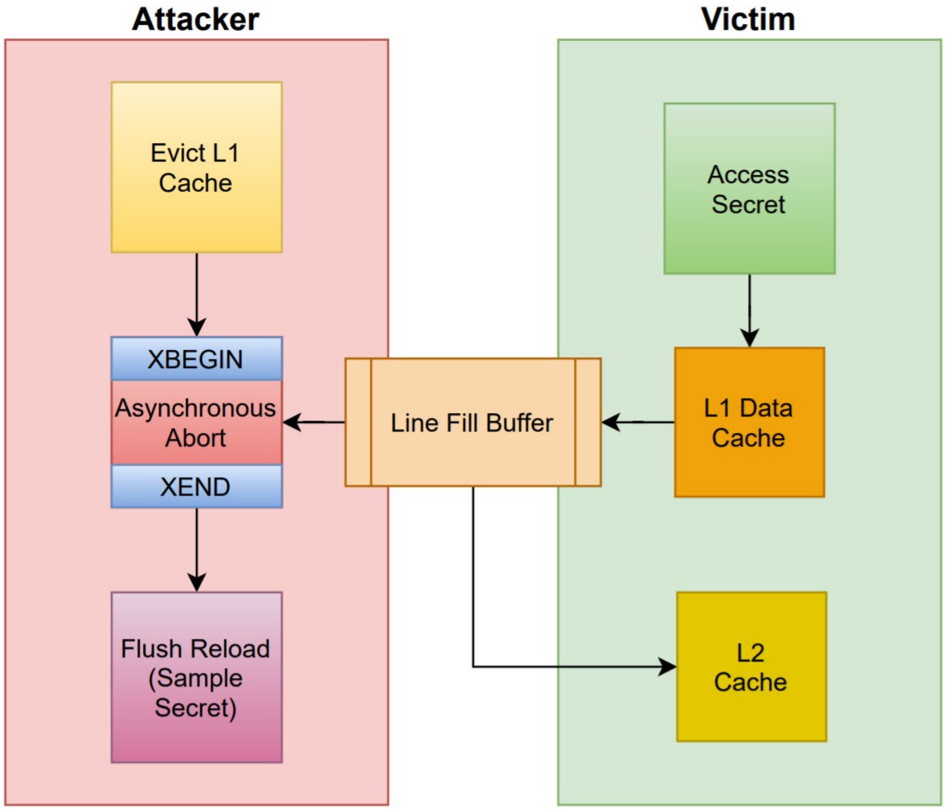
Intel Hyper-threading



Asynchronous Aborts



TAA Vulnerability



Related Papers & Existing mitigation

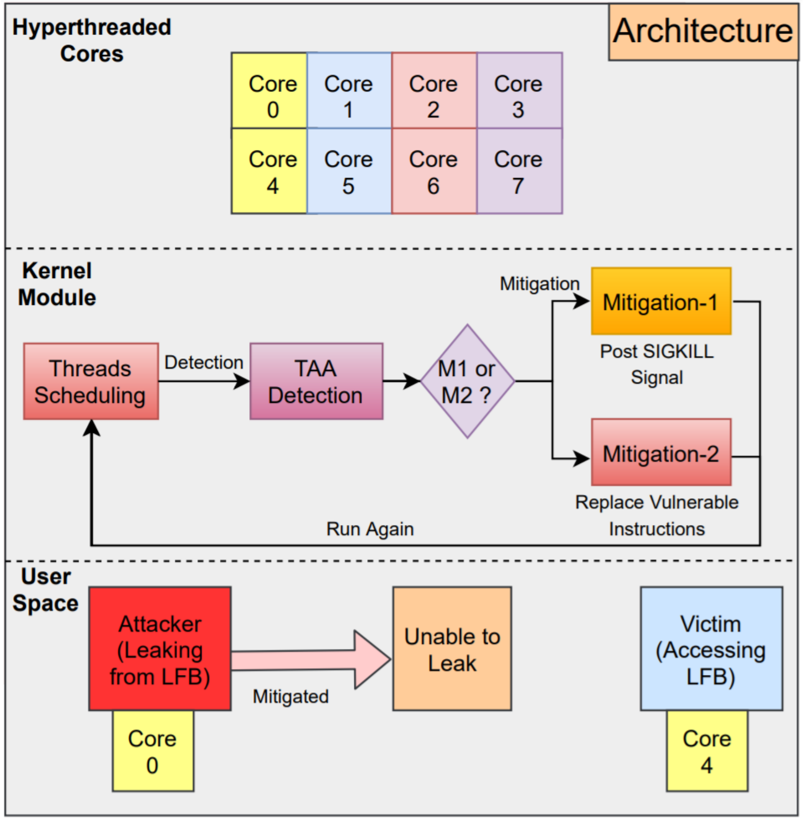
- CacheOut, RIDL and Zombieload
- VERW mitigation
- No solution for hyperthreaded TAA attacks

Contributions

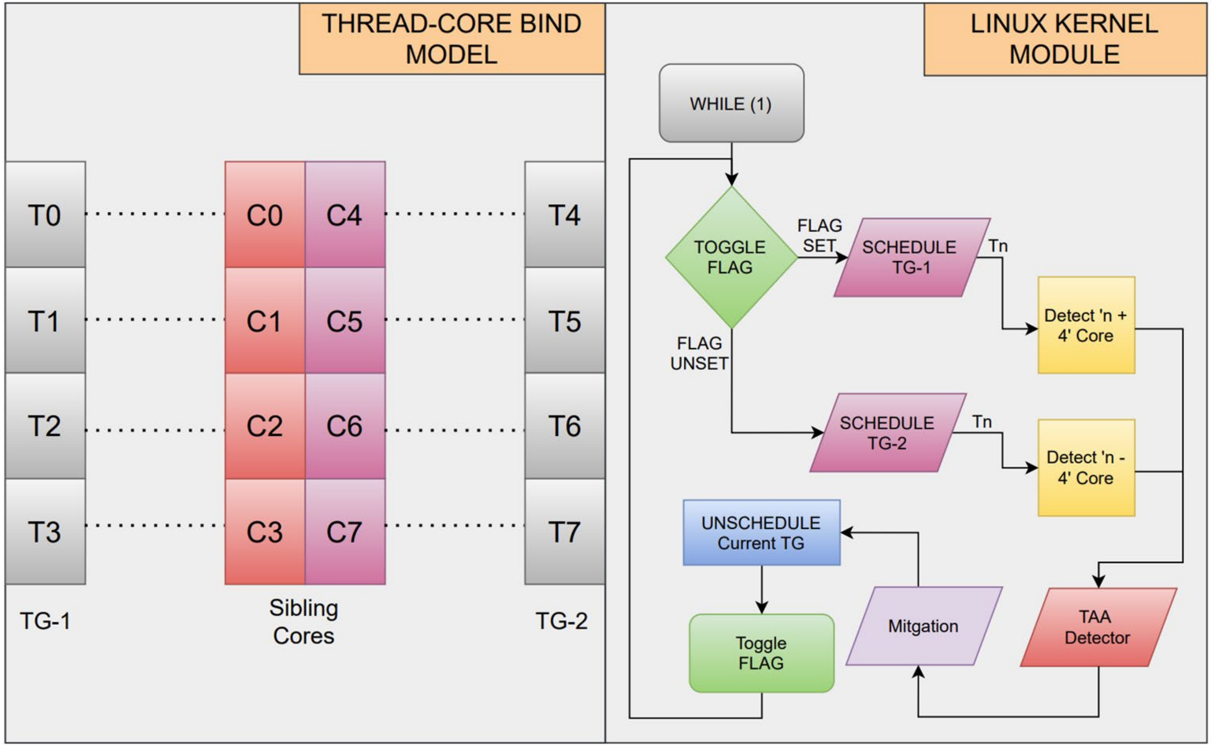
- Countermeasure for TAA vulnerability
- Hyperthreaded & Cross-cores TAA attacks
- Efficient solution
- Scalable solution
- Noise resilient

Methodology

Methodology



Scheduling


THREAD-CORE BIND MODEL
T0
T1
T2
T3
C0
C1
C2
C3
C4
C5
C6
C7
T4
T5
T6
T7
TG-1
Sibling Cores
TG-2
LINUX KERNEL MODULE
WHILE (1)
TOGGLE FLAG
FLAG SET
SCHEDULE TG-1
Tn
Detect 'n + 4' Core
FLAG UNSET
SCHEDULE TG-2
Tn
Detect 'n - 4' Core
UNSCHEDULE Current TG
Toggle FLAG
Mitgation
TAA Detector

Detection

- Analyzing cache conflicts
- Feature Selection
- Threshold based Detection

Detection: Feature Selection

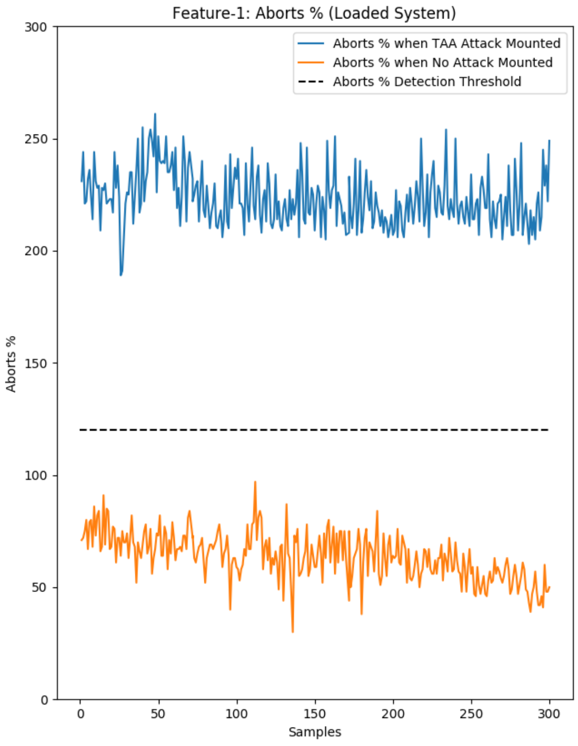
- Feature-1
  - Total number of cache conflicts
- Feature-2
  - Maximum number of cache conflicts occurred on a cache set
- Feature-3
  - Minimum number of cache conflicts occurred on a cache set

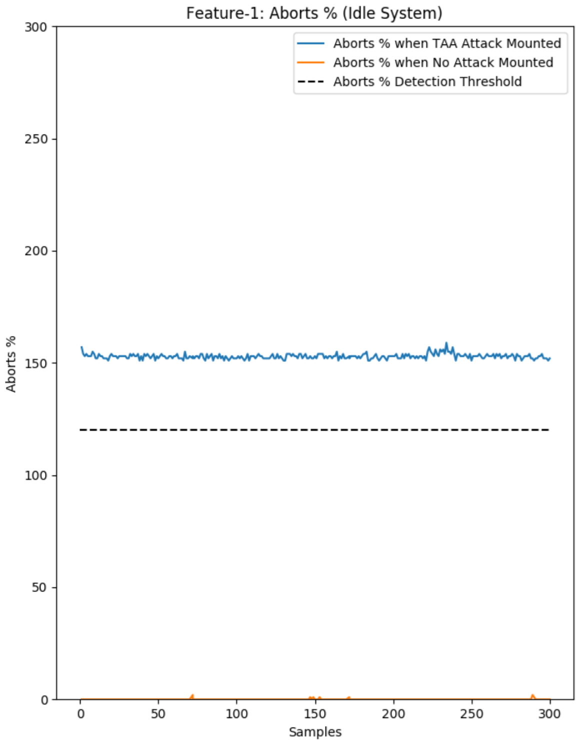
Detection: Algorithm

Algorithm 1: TAA Attack Detection

```
abrt ← 0, no_abrt ← 0, abrt_per_set[SETS] ← 0, samples ← 0
while samples ≤ 50000 do
    for set ← 0 to 64 by 1 do
        Begin_TSX_Transaction();
        Access_All_Ways(set);
        End_TSX_Transaction();
        if ABORT_REASON_CACHE_CONFLICT then
            abrt ← abrt + 1;
            abrt_per_set[set] ← abrt_per_set[set] + 1;
        else if NO_ABORT then
            no_abrt ← no_abrt + 1;
    samples ← samples + 1;
    Sleep_Detection();
if ((abrt / no_abrt) * 50000 ≥F1_THRESHOLD &&
Get_Max(abrt_per_set) ≥F2_THRESHOLD &&
Get_Min(abrt_per_set) ≥F3_THRESHOLD) then
    /* Attack Detected */
    return 1;
/* No-Attack Detected */
return 0;
```

Detection: Threshold Calculation





Mitigation

- Mitigation-1 (SIGKILL to Process):
  - Posts SIGKILL signal from kernel to Vulnerable process to kill it
- Mitigation-2 (Instruction Replacement):
  - Replaces vulnerable instruction that causes the attack within Linux Kernel

Mitigation: SIGKILL to Vulnerable Process

Algorithm 2: Mitigation-1 (SIGKILL to Vulnerable Process)

```
if Is_TAA_Detected() then
    task_struct ← Get_Task_Struct();
    Task_Lock(task_struct);
    ret ← Send_SIG_KILL_Signal(task_struct);
    Task_Unlock(task_struct);
    if ret == SUCCESS then
        /* Killed vulnerable process */
        return 1;
    /* Unable to kill vulnerable process */
    return 0;
```

Mitigation: Vulnerable Instructions Replacement

Algorithm 3: Mitigation-2 (Vulnerable Instruction Replacement)

```
if Is_TAA_Detected() then
    task_struct ← Get_Task_Struct();
    Task_Lock(task_struct);
    user_pages ← Read_Text_Section(task_struct);
    text_ptr ← Map_Kernel_Space(user_pages);
    for i ← 0 to code_size by 1 do
        if text_ptr[i] == VULNERABLE_INSTRUCTION then
            Replace_To_NOP(text_ptr[i]);
            ret ← SUCCESS;
    Unmap_And_Release_Pages(user_pages);
    Task_Unlock(task_struct);
    if ret == SUCCESS then
        /* Replaced Vulnerable Instruction */
        return 1;
    /* Unable to find Vulnerable Instruction */
    return 0;
```

Results

Experimental Results

System State	Type	Accuracy (%)	FP (%)	FN (%)	Overhead (%)	Latency (us)
IDLE	**Detection**	97.31	2.64	0.03	2.5	5264232
	Mitigation-2	99.94	0.03	0		306
	Mitigation-1	99.85	0.03	0.18		86
LOADED	**Detection**	98.26	1.73	0.03	2.5	6916110
	Mitigation-2	99.91	0.03	0.06		452
	Mitigation-1	99.82	0.03	0.21		106

Conclusion

- Countermeasure for TAA Vulnerability
- Works for both cross & hyper-threaded cores
- High accuracy, low overhead and low latency
- Standalone Mitigation
- Scalable solution