Speculative Execution Attacks

Background Branch Instructions

- Branch instructions
  - Instructions that change the flow of execution
  - Call, return, if, while, for, goto
- Branch Instructions require time to decode
- Conditional Branches
  - May or may not be taken
  - If, while
- Unconditional Branches
  - Will be taken
  - Call, return, goto

```
string inputString;

cout << "Enter student names (max of 50).  Enter q to quit" << endl;

int count = 0;
bool continueInput  = true;

while (count ++ < 50 && continueInput)
{
        cout << "Name: ";
        cin >> inputString;
        if (inputString == "q")
                continueInput = false;
}
```

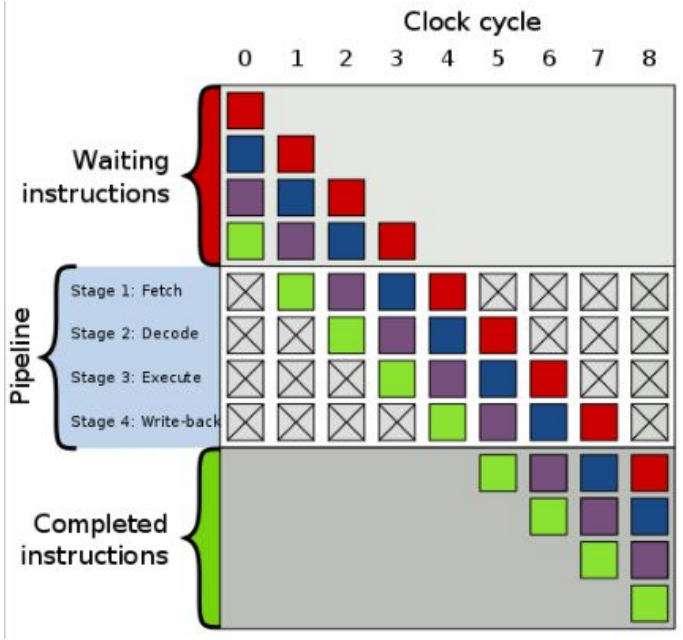
Background Branch Predictors

Branch Predictors predict:

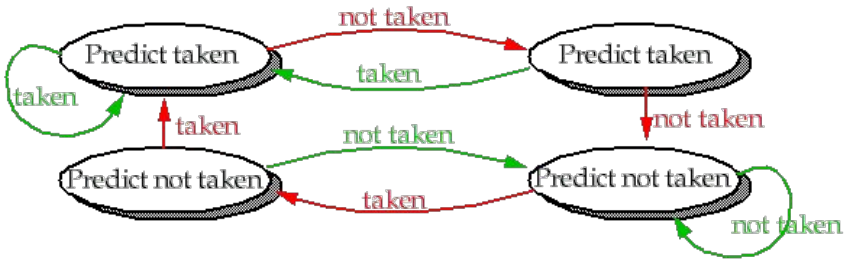
1. Where execution will go (target address)
2. Whether a branch (conditional) will be taken

BPs use history to determine prediction

Basic 2 bit branch predictor
Dynamically update prediction



	Bit 1 - 0	Bit 1 - 1
Bit 2 - 0	Not taken	Not taken
Bit 2 - 1	Taken	Taken



Background: Misprediction?

What happens after a branch prediction is made?

Run in “Speculative Execution”

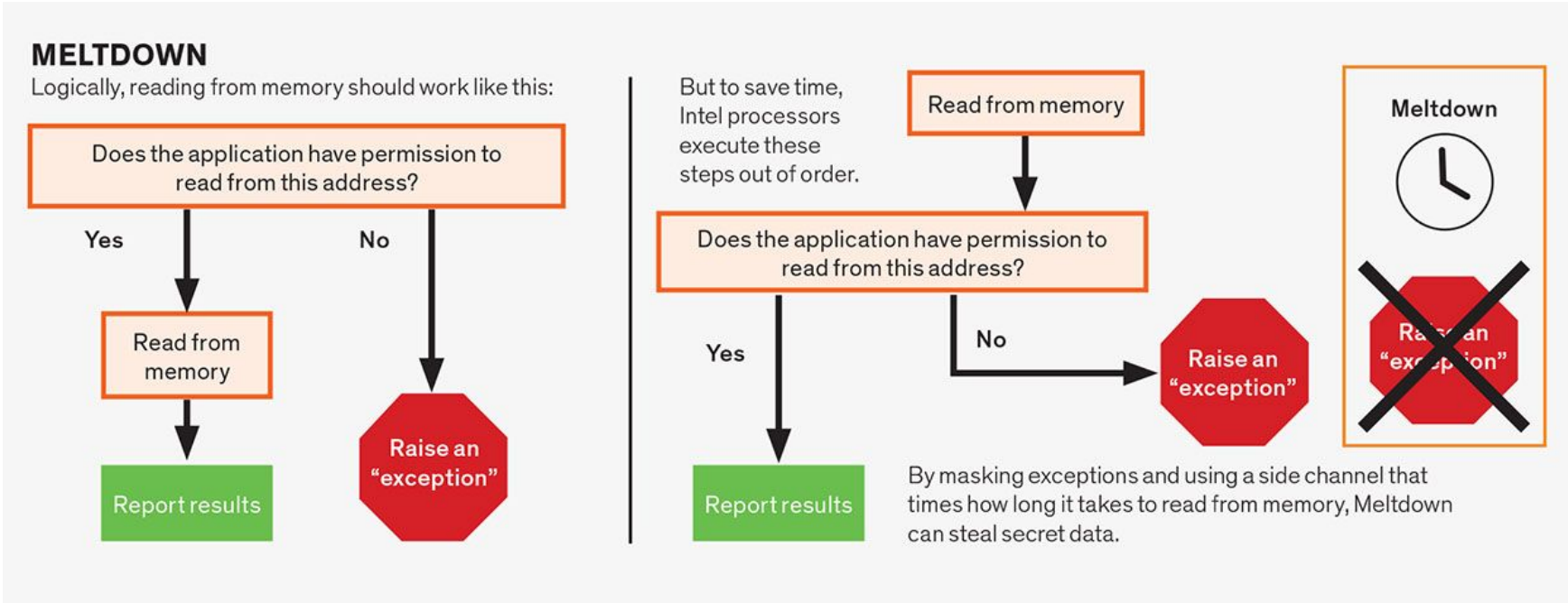
1. Save state of processor (registers, data, stack point, instruction point)
2. Then run in speculative execution
3. If prediction is correct - Continue execution
4. Otherwise reset state of processor before misprediction

Mispredictions are costly as they reset the microarchitectural state

But not completely…

Meltdown


Meltdown


MELTDOWN
Logically, reading from memory should work like this:
Does the application have permission to read from this address?
Yes
No
Read from memory
Report results
Raise an "exception"
But to save time, Intel processors execute these steps out of order.
Read from memory
Does the application have permission to read from this address?
Yes
No
Raise an "exception"
Report results
Meltdown
By masking exceptions and using a side channel that times how long it takes to read from memory, Meltdown can steal secret data.

Spectre V1

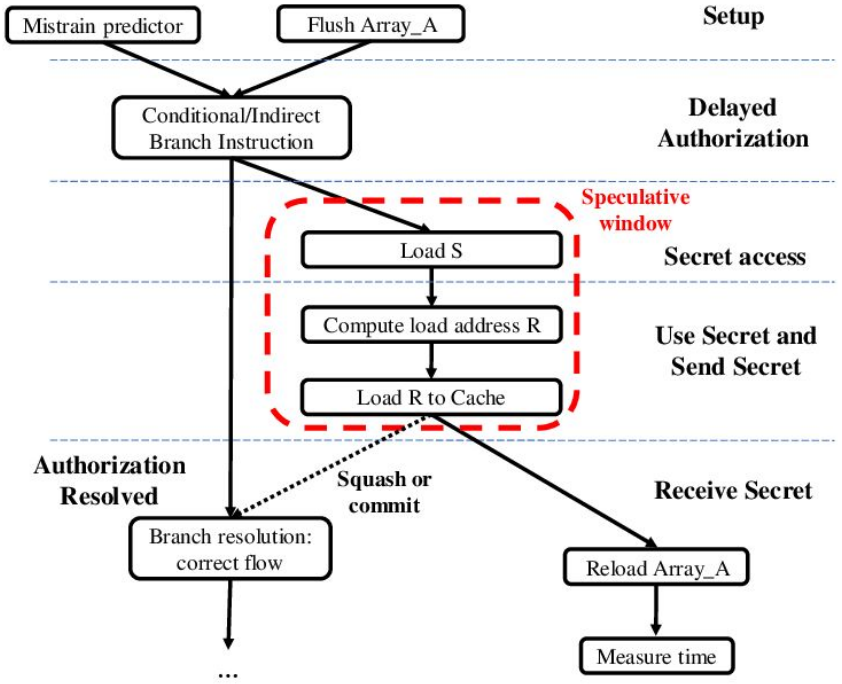
```
if (x < array1_size)
    y = array2[array1[x] * 256];
```

Takes advantage of out of order checks

In speculative execution processor waits to perform authorization check until after speculative execution window

Allows attackers to read data they do not have access to (loads into cache)

Once data is in cache perform a cache side channel attack to read the data



Spectre v2

1. Attacker mistrains branch target
    a. Allows attacker to point branch at their code
    b. Attacker sets a “payload”
2. Victim runs program
3. Branch predictor mispredicts branch and execution starts at payload
4. Data from payload is loaded into cache
5. Attacker performs cache side channel to read data

Impact

	Meltdown	Spectre
Architecture	Intel, Apple	Intel, Apple, ARM, AMD
Entry	Must have code execution on the system	Must have code execution on the system
Method	Intel Privilege Escalation + Speculative Execution	Branch prediction + Speculative Execution
Impact	Read kernel memory from user space	Read contents of memory from other users' running programs
Action	Software patching	Software patching (more nuanced)

Daniel Miessler 2018

Protections

- Meltdown is fixed with simple software update
- Spectre v1 and v2 are more difficult to protect against
- Protections are costly (worse performance)
- AMD recently released update (March 12th, 2022 to help protect against Spectre v2 attacks)

STBPU

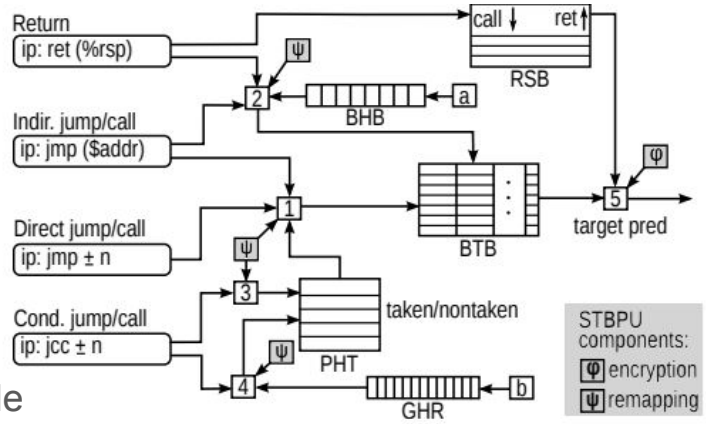
Motivation and Idea

- Software fixes have significant performance costs
- Current fixes are targeting results of a problem not the root of the problem itself
- Want to make a branch predictor unit (BPU) that is secure and prevents programs from interfering with each other
- Use secret tokens to to prevent branches from interfering with each other

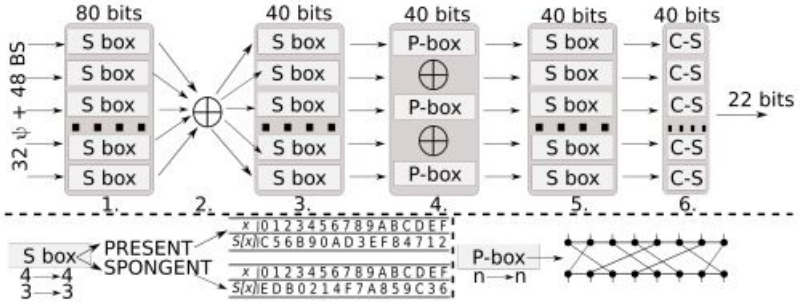
STBPU Threat Model

- Powerful attack, complete understanding of underlying hardware
- Cannot access ST
    - Stored in speical register that requires privileged mode to read
    - When stored by system software, it can only be read when complete system compromise
        - In this case attacker has no need for side channels
        - Bigger problems
- Spectre v1 not in scope, Spectre v2 is in scope
    - Different problems to fix each, require different fixes
- Two models
    - Sensitive Process as a victim
    - Kernel as the victim

STBPU Design: Secret Tokens



- Each process has a secret token (ST)
- STs are used in two places
  - Firstly to encrypt (and decrypt) entries in the BPU table
  - Secondly to be added to remapping functions to make them dependent on the ST
- Normal BPU
  - Uses a remapping function to look inside their lookup table to get prediction
  - Then sends prediction (both if taken and target address)
- STBPU
  - Remapping lookup is dependent on ST
  - Prediction result is encrypted with ST



STBPU: Security Evaluation

- In STBPU if attacker mistrains a branch
  - Victim runs that branch it will map to a different place in prediction table
  - Each process (victim and attacker) will have a different ST that maps to a different location in BPU
- In STBPU if attacker manages to have a branch collision (same place in lookup table)
  - Decoded branch will be different as each branch target is encrypted with ST of that process

	Baseline input	STBPU input	Output	Function
1	32 s	32 ψ, 48 s	9 ind, 8 tag, 5 offs	$R_1(80 \mapsto 22)$
2	58 BHB	32 ψ, 58 BHB	8 tag	$R_2(90 \mapsto 8)$
3	32 s	32 ψ, 48 s	14 ind	$R_3(80 \mapsto 14)$
4	18 GHR, 32 s	32 ψ, 16 GHR, 48 s	14 ind	$R_4(96 \mapsto 14)$
t	48 s, L(GHR)	32 ψ, 48 s, L(GHR)	10/13 ind, 8/12 tag	$R_t(80\uparrow \mapsto 25)$
p	48 s	32 ψ, 48 s	10 ind	$R_t(80 \mapsto 10)$

L(GHR) — represents geometric series of global history lengths

s — represents the source bits of branch instructions

ST Rerandomization

- Even with ST attacker can brute force to find ST
- OS keeps track of mispredictions and eviction from BPU. After a certain amount the OS gives new STs to all processes
  - Note information encoded in branch predictor (ie target addresses that are encrypted) are not reencoded
  - They will remain with previous encryption, however if we try to decrypt it, it will be with new ST so result will be different
- Too frequent rerandomization can affect performance
- Too infrequent rerandomization can harm security

Performance Evaluation

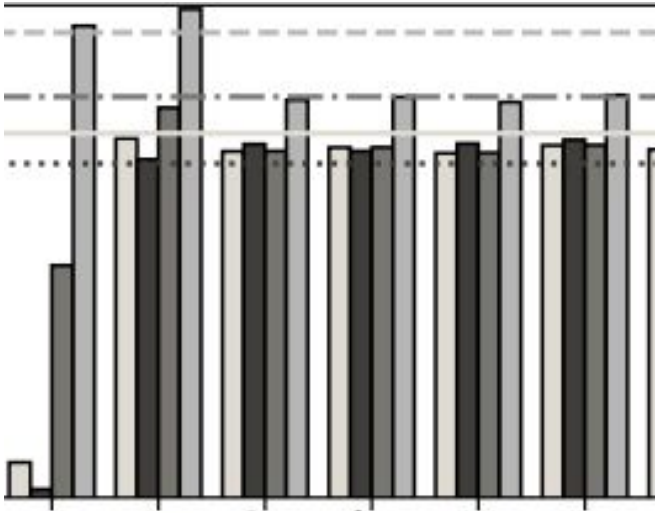
Trace testing to test against other models of secure BPUs

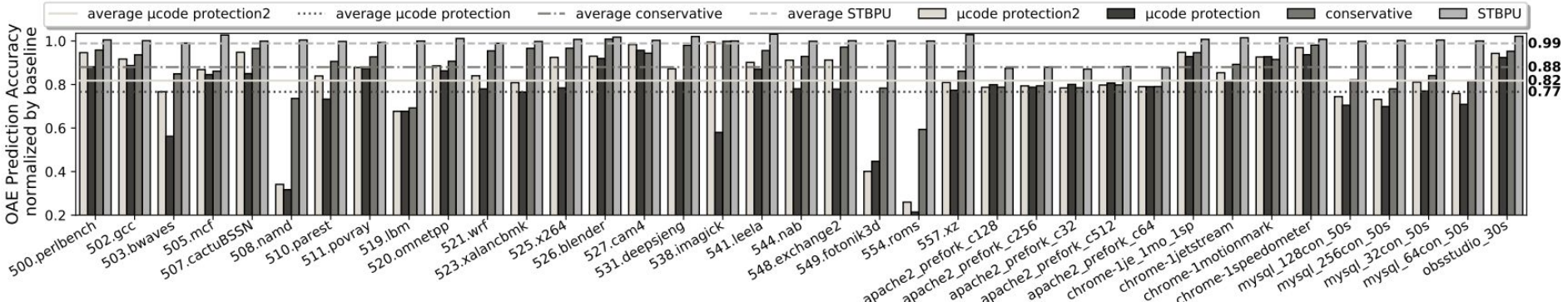
Used Gem5 to perform testing of STBPU model

- Evaluated STBPU implementation on 4 BPUs
  - Basic Branch Predictor
  - Perceptron Branch Predictor
  - TAGE Branch Predictor 8KB
  - TAGE Branch Predictor 64KB
- Used branch prediction accuracy and IPC (instructions per cycle) to measure performance vs baseline models

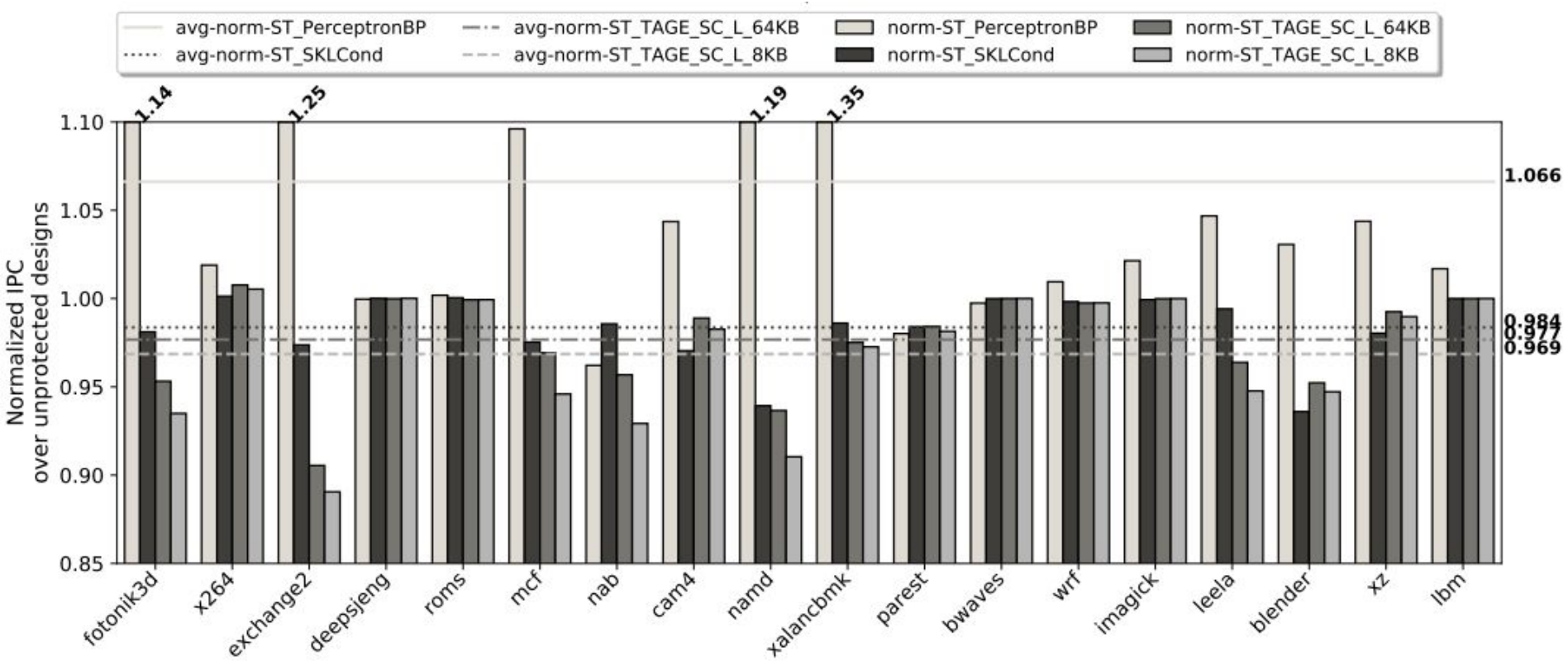
Accuracy vs Secure BPU Models

- STBPU - 99% accuracy
- Conservative - 88%
- Ucode - 77%
- Ucode2 - 82%





Gem5b IPC


avg-norm-ST_PerceptronBP
avg-norm-ST_SKLCond
avg-norm-ST_TAGE_SC_L_64KB
avg-norm-ST_TAGE_SC_L_8KB
norm-ST_PerceptronBP
norm-ST_SKLCond
norm-ST_TAGE_SC_L_64KB
norm-ST_TAGE_SC_L_8KB
Normalized IPC
over unprotected designs
1.10
1.05
1.00
0.95
0.90
0.85
1.14
1.25
1.19
1.35
1.066
0.984
0.977
0.969
fotonik3d
x264
exchange2
deepsjeng
roms
mcf
nab
cam4
namd
xalancbmk
parest
bwaves
wrf
imagick
leela
blender
xz
lbm

Threshold Randomization Testing

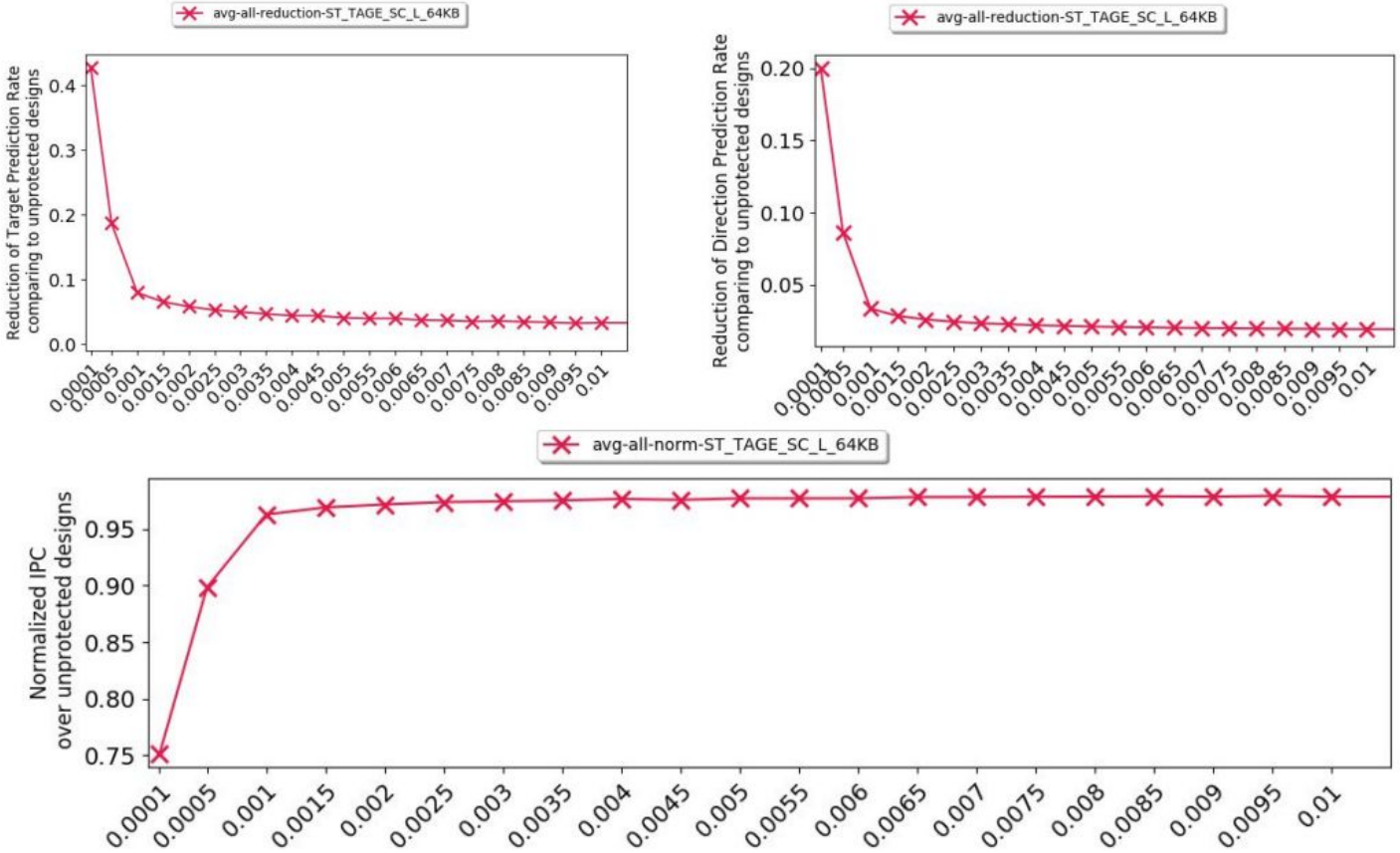


Image Sources

https://meltdownattack.com/

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9407076

https://spectrum.ieee.org/how-the-spectre-and-meltdown-hacks-really-worked