

# RSA<sup>®</sup>Conference2020

San Francisco | February 24 – 28 | Moscone Center

**HUMAN**  
ELEMENT

SESSION ID: HTA-T12

## Nowhere to Hide: How HW Telemetry and ML Can Make Life Tough for Exploits



**Dr. Zheng Zhang**

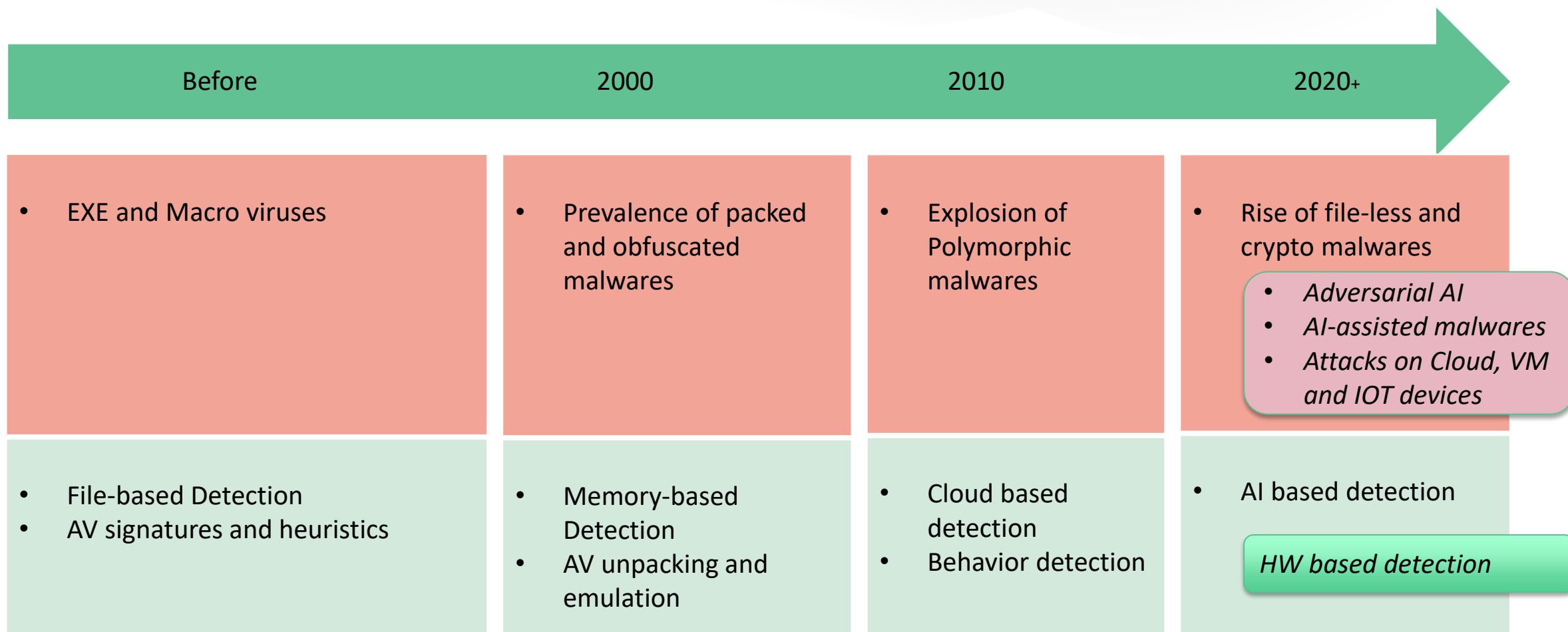
Principal Engineer  
Intel Corporation

**Rahul Ghosh**

Security Architect  
Intel Corporation

#RSAC

# The Evolution of Malware Detection Technologies



# CPU Telemetry To The Rescue...wait, what is it?

- Modern processors have on-chip units to monitor microarchitectural events at runtime.
- Mix of architectural (e.g. core cycles, instructions retired) and non-architectural events (e.g. TLB refs, L1/L2 refs, resource stalls...) available depending on CPU sku and vendor
- Can be sampled based on time (e.g. every n msecs) or event count (e.g. every n counts of an event).
- Typically 4-8 events can be monitored at a time per logical core programmed with Model Specific Registers (MSRs).  
CPU generation and vendor dependent.

**RSA**®Conference2020

# Threat Detection

Detecting malware at runtime by profiling with CPU telemetry

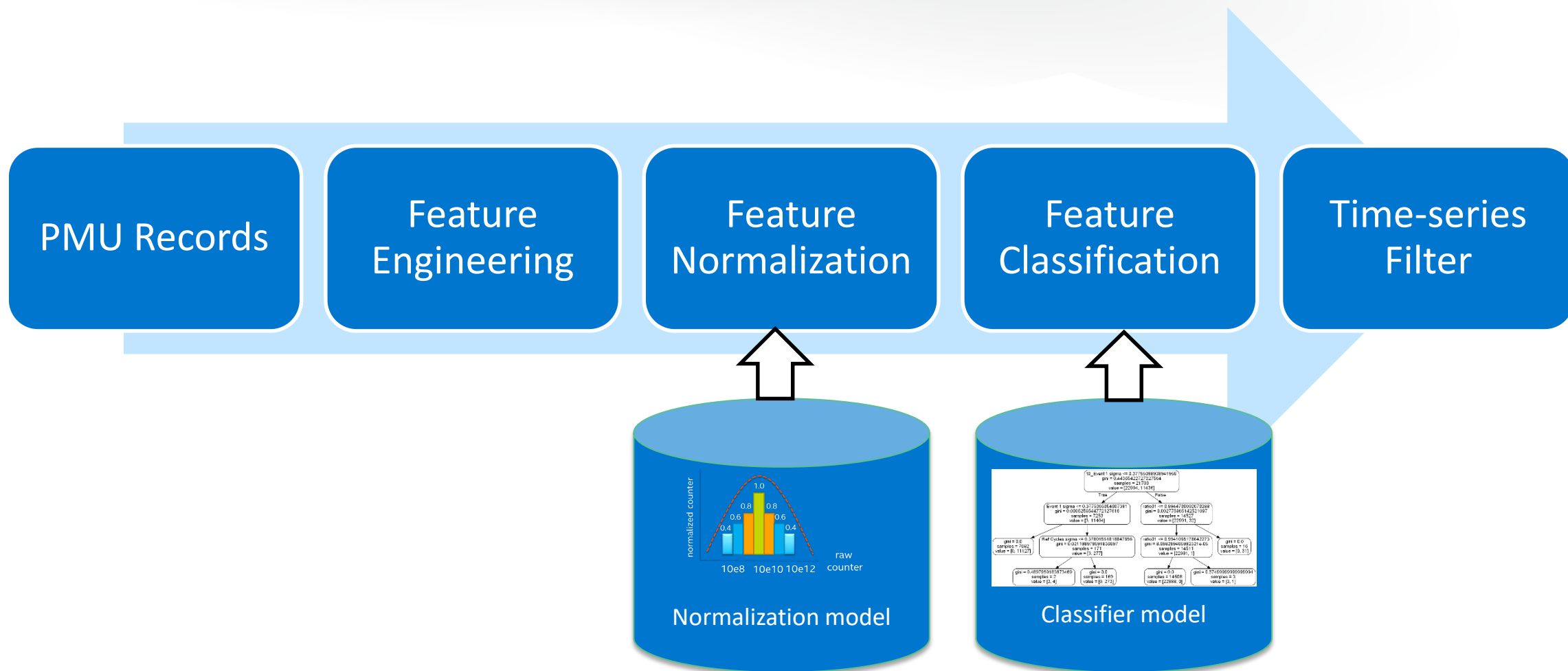
# Profiling exploits with performance monitoring events

- Exploit behaviors suitable for CPU profiling
  - repetitive computations with sustained execution behavior
  - E.g. hashing, encryption, heap spraying, cache manipulation...
- Exploit execution affects specific performance monitoring (perfmon) events more than others. Different for each malware class.
- The correlation of this set of Relevant Events (REs) constitutes that exploit's 'telemetry signature'.
- The more REs we can identify the more distinct the exploit's telemetry signature is from the multitude of benign workloads
- Signature matching becomes a classification problem solved with ML.

# DEMO



# Classification Pipeline



# Which PMU Events?

- For example purposes, Intel's 9<sup>th</sup> gen Core CPUs have
  - ~800 core events
  - ~300 uncore events
  - > 10 events related to L3 cache misses
  - > 15 events related to DTLB load misses
  - > 20 events related to resource stalls
  - ....
- How do we know what is relevant?



See <https://download.01.org/perfmon> - list of perfmon events for most Intel CPU generations



# Information Gain is the key

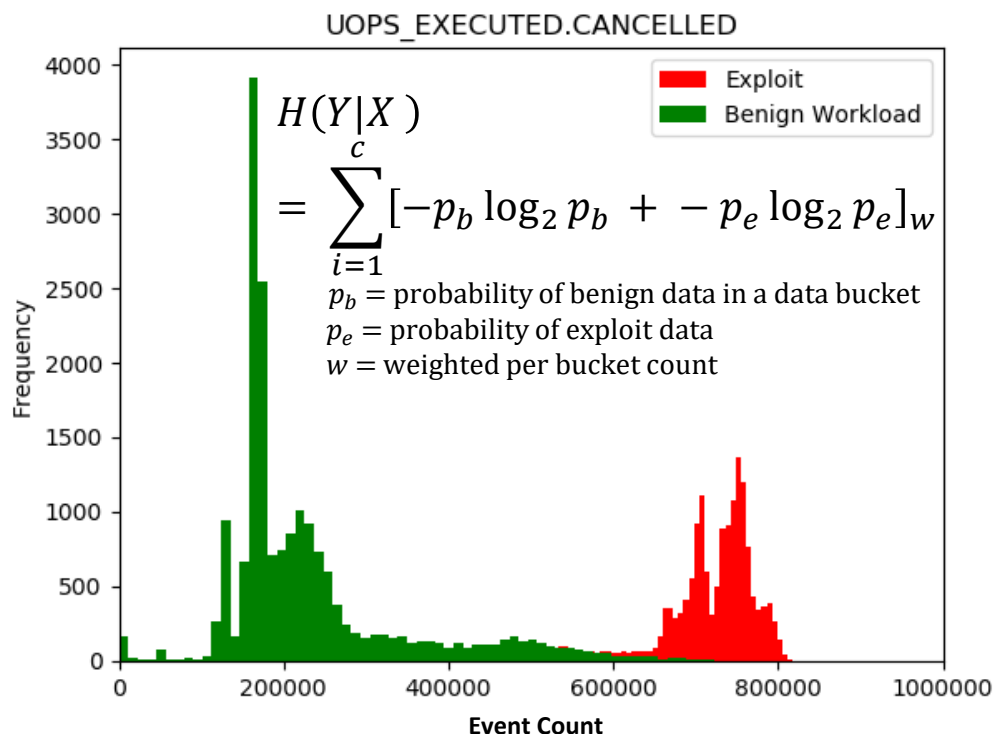
- Information Gain (IG) is based on the entropy of an event provides a measure of its relevancy in profiling an exploit.
- A Relevant Event needs to have significantly lower entropy with the exploit than without.
- $IG(Y, X) = H(Y) - H(Y|X)$  where  
     $H(Y)$  – entropy without knowledge of event values  
     $H(Y|X)$  – entropy with knowledge of event values.  
     $X$  represents the event values.
- Select  $n$  events with the highest IG for the exploit class.  
     $n=3-5$  is typically sufficient.

# Sample Telemetry

Timestamp	Process ID	Thread ID	Core ID	Ref Cycles	Event 0	Event 1
4429226900489	2587	2601	1	16981406	1001104	449148
4429230923370	2114	2131	3	9372032	1000378	360467
4429232849307	2114	2131	1	18745524	999254	511086
4429237263792	2114	2131	0	18833708	1000215	409082
4429241873848	2114	2131	2	58597246	1000248	551597
...						

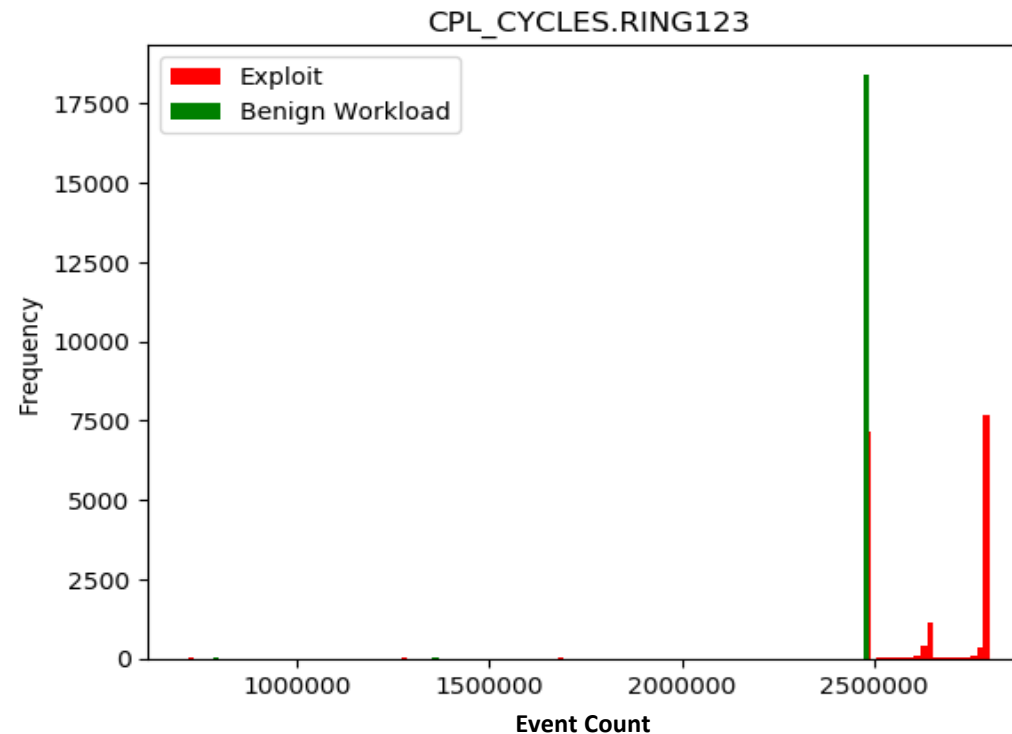
Data was collected using event based sampling from a 4 core system

# Identifying Relevant Events



Separation in event count buckets between exploit and benign workloads.  $H(Y|X)$  will be low, hence high IG.

**Ideal candidate.**



Some unique exploit buckets but still significant overlap in some buckets between exploit and benign workloads.  $H(Y|X)$  will be higher, hence lower IG.

**Not an ideal candidate.**

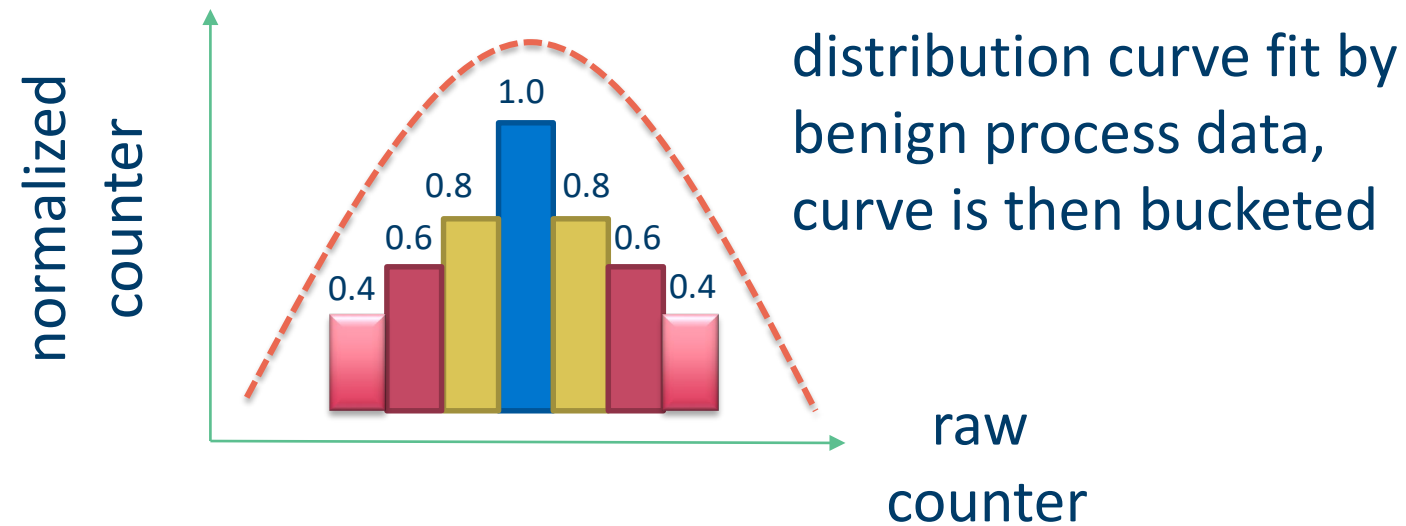
## Relevant Events → Features

- For a given event value ( $v$ ) and the feature histogram ( $h$ ) we can compute the following probability values:
  - ❖ Probability density function  $pdf(h, v) = p(x = v|h)$
  - ❖ Cumulative distribution function  $cdf(h, v) = p(x \leq v|h)$
  - ❖ Sigmoid function  $sigma(h, v) = sigmoid\left(\frac{v - \bar{v}}{v_{max} - v_{min}}\right) | h$

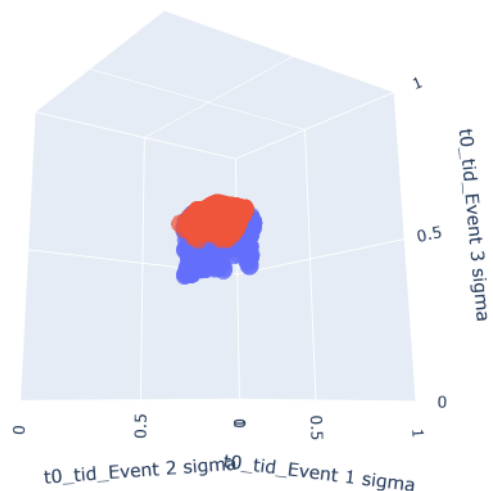
# Feature Normalization

- Likelihood of a feature mapping to normal system behavior
- Can either be trained offline or online on the target system

Ref Cycles pdf	Ref Cycles cdf	Ref Cycles sigma	Event 1 pdf	Event 1 cdf	Event 1 sigma	t0_Event 1 pdf	t0_Event 1 cdf	t0_Event 1 sigma	Label
1	0	0.3785	1	0.000502	0.378539	0	0	0.377492	1
1	0	0.37854	1	0.000502	0.378728	1	0	0.37756	1
1	0	0.378368	1	0.000502	0.378483	1	0	0.377616	1

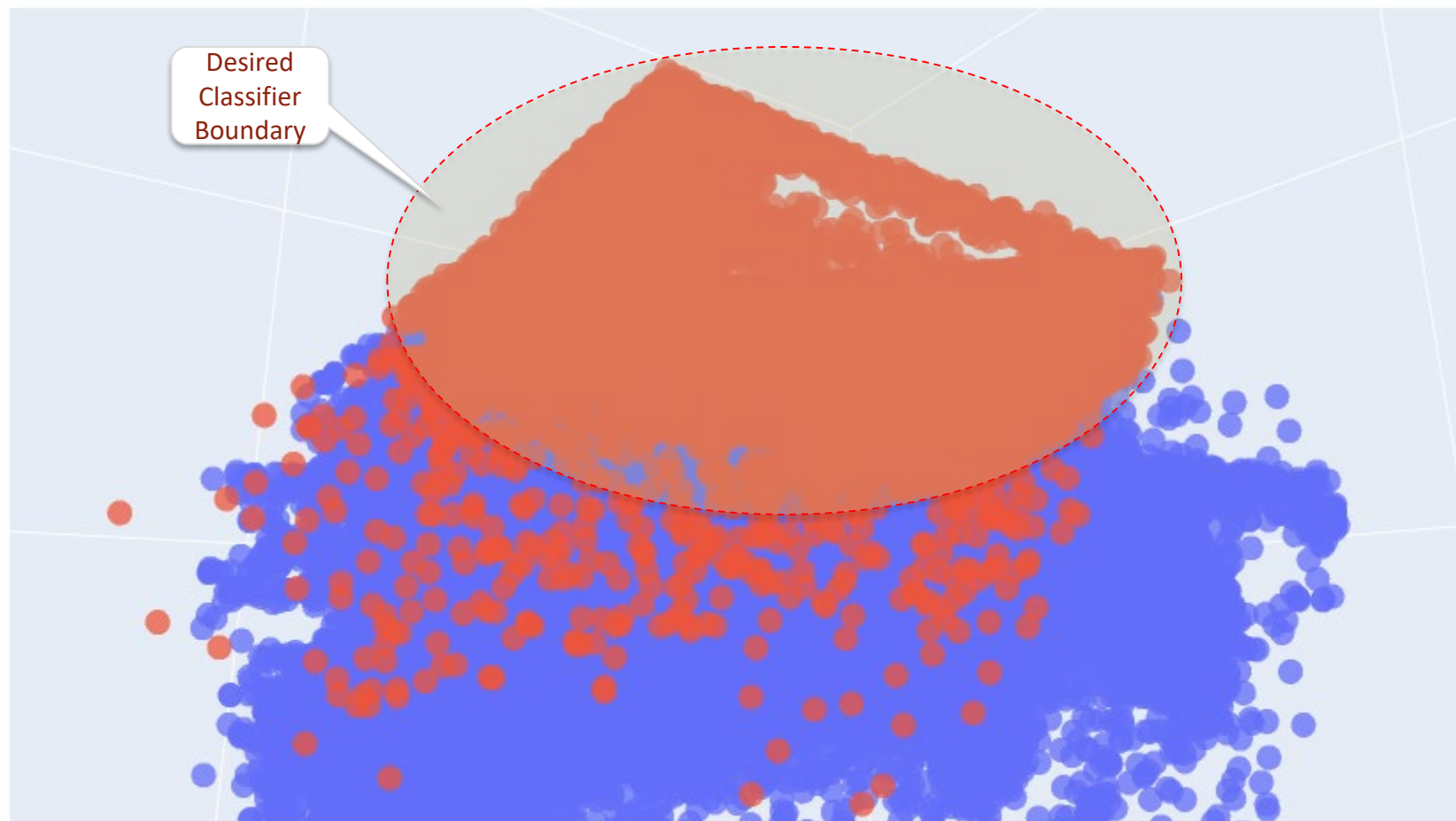


# Feature Map Example



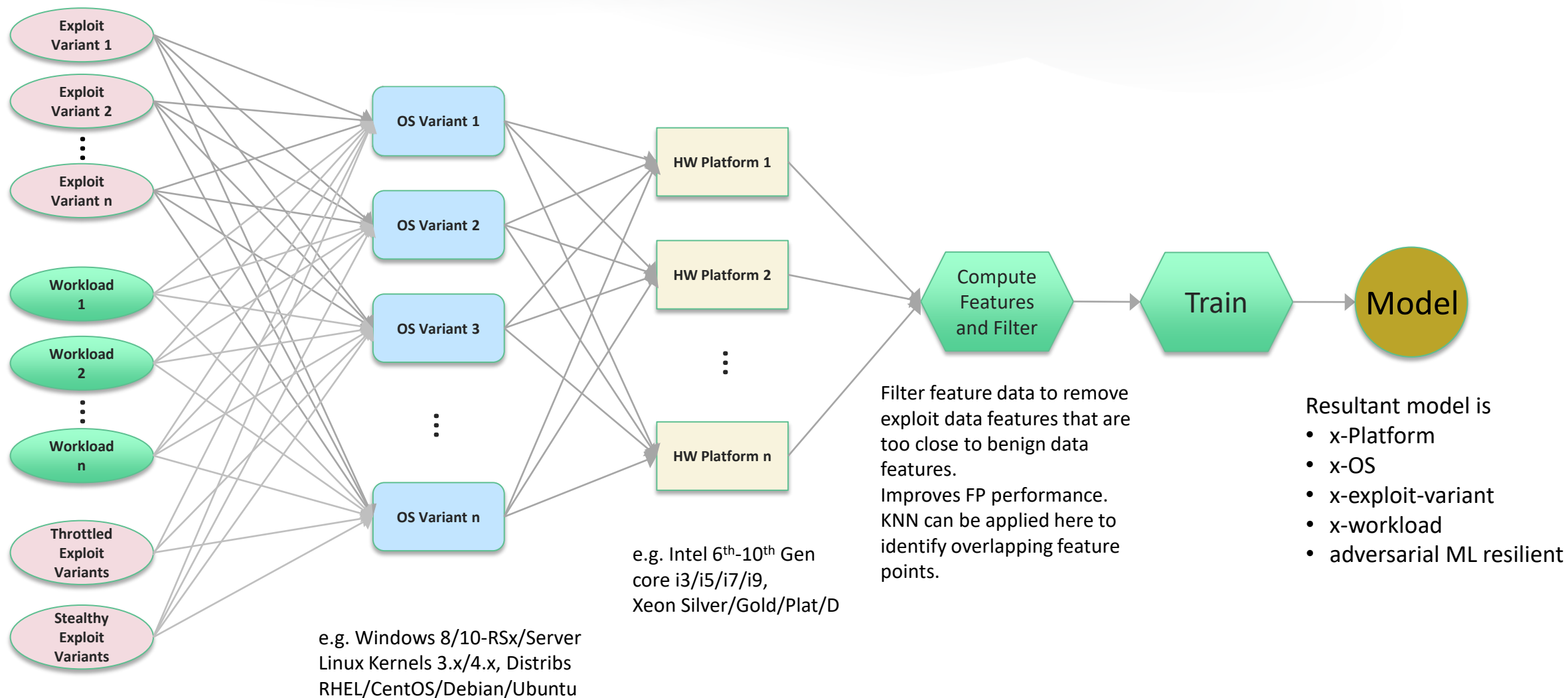
Benign Workloads/Apps

Exploits



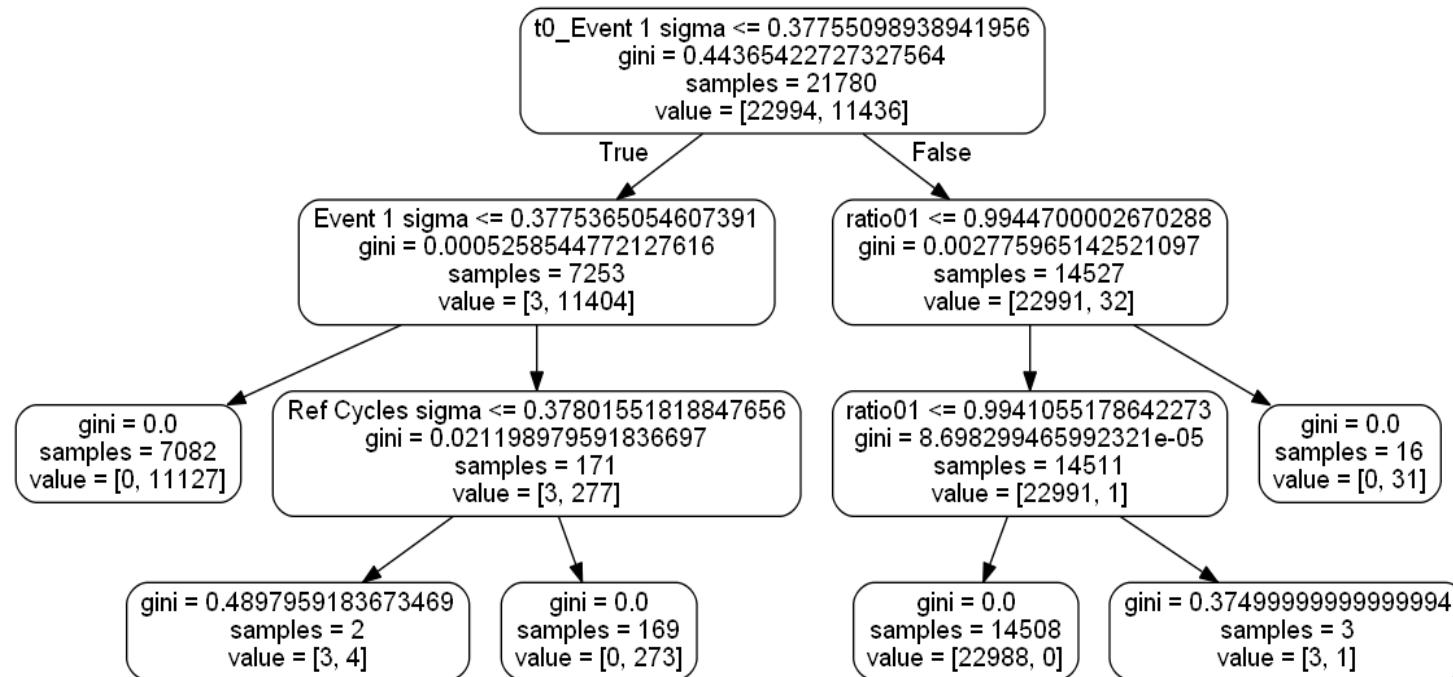
3D feature map of a cryptocurrency miner showing distinct feature space from typical benign apps. Data in this example was collected from multiple Intel platforms running Windows OS.

# Training pipeline



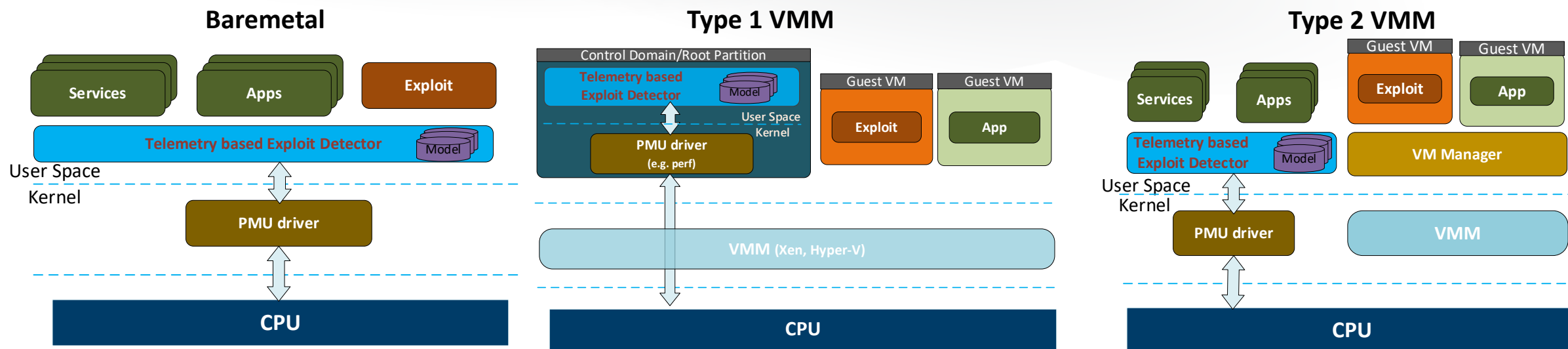
# Classification

- Linear independence not guaranteed in CPU telemetry
- Non-linear classifiers worked better - SVM, LSTM, Decision Trees, Ensemble learning.
- Best results with a Random Forest classifier in our experiments.





# Detection Environments



- Profiling and detection possible in multiple deployment environments
- Baremetal and Type-2 VMM are typically set up with access to guest VM telemetry.
- Some Type 1 VMMs like VMWare and Hyper-V needs special configuration for access to guest VM telemetry from Dom-0.
- In virtualized environments training data should include exploits and workloads running within VMs for better results.

# What else can we use from the CPU

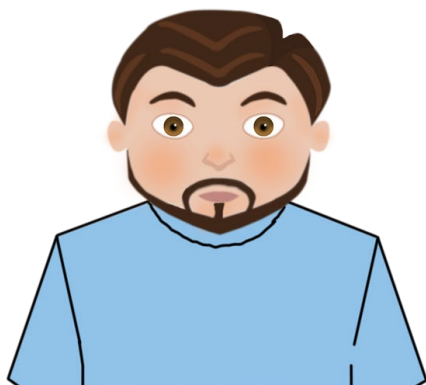
- Precise Event Based Sampling (PEBS)
  - Extension of sampling based event collection
  - Additional data like Instruction Pointer (IP), Flags, Registers, transactional memory data, precise TSC value.
  - Adds some overhead but metadata can be used for fine grained profiling.
- Last Branch Records (LBRs)
  - Records 4-32 branches executed depending on CPU generation
  - Can be configured to record the call stack
  - Additional info like successfully predicted branch

**RSA**®Conference2020

# Anomaly Detection

Detecting unknown malware at runtime with CPU telemetry

# Signature Detection



Pass

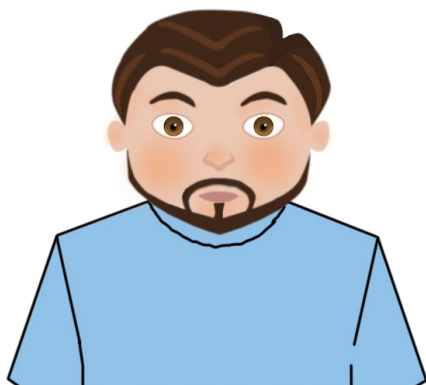


Jail



Pass

# Anomaly Detection



Pass



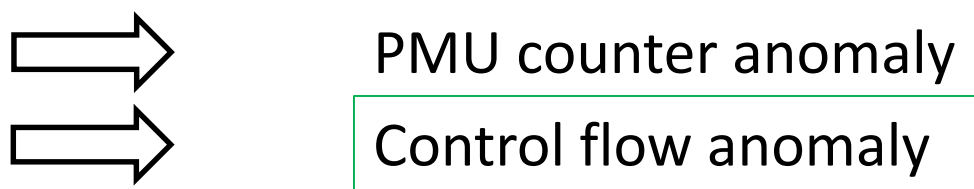
Block



Block

# HW-based Anomaly Detection

- Why HW telemetry?
  - Capable of characterizing the CPU and instruction level program behaviors
  - Less susceptible to common evasion techniques
  - Rich and diverse data sources for ML/AI modeling
  - Complementary to SW telemetry
- What telemetry?
  - PMU events
  - Instruction Traces



PMU counter anomaly

Control flow anomaly

# What is Control Flow?

- Control flow is the order in which branch instructions are executed.

```
.text:00000000000008F4      lea     rdi, aLoadingDlls___ ; "Loading DLL '%s' ... "
.text:00000000000008FB      mov     eax, 0
.text:0000000000000900      call   _printf
.text:0000000000000905      mov     rax, [rbp+file]
.text:0000000000000909      mov     esi, 2             ; mode
.text:000000000000090E      mov     rdi, rax           ; file
.text:0000000000000911      call   _dlopen
.text:0000000000000916      mov     [rbp+handle], rax
.text:000000000000091A      cmp     [rbp+handle], 0
.text:000000000000091F      jz      short loc_92A
.text:0000000000000921      lea     rax, aOk           ; "OK"
.text:0000000000000928      jmp     short loc_931
.text:000000000000092A      ; -----
.text:000000000000092A      loc_92A:      lea     rax, aFailed       ; CODE XREF: main+75↑j
.text:000000000000092A      ; "FAILED"
.text:0000000000000931      loc_931:      mov     rdx, [rbp+handle] ; CODE XREF: main+7E↑j
.text:0000000000000931      mov     rsi, rax
.text:0000000000000935      lea     rdi, aSP           ; "%s (%p)\n"
.text:0000000000000938      mov     eax, 0
.text:000000000000093F      call   _printf
.text:0000000000000944      cmp     [rbp+handle], 0
.text:0000000000000949      jnz     short loc_973
.text:0000000000000950      call   _dLError
.text:0000000000000955      mov     rdx, rax
.text:0000000000000958      mov     rax, cs:stderr@GLIBC_2.2.5
.text:000000000000095F      lea     rsi, aLoadErrorS ; "Load Error: %s\n"
.text:0000000000000966      mov     rdi, rax           ; stream
.text:0000000000000969      mov     eax, 0
.text:000000000000096E      call   _fprintf
.text:0000000000000973      loc_973:      cmp     [rbp+handle], 0 ; CODE XREF: main+A4↑j
.text:0000000000000978      ;
```



GLIBC!printf

...

.text:0000000000000905

GLIBC!dlopen

...

.text:0000000000000916

GLIBC!printf

...

.text:0000000000000949

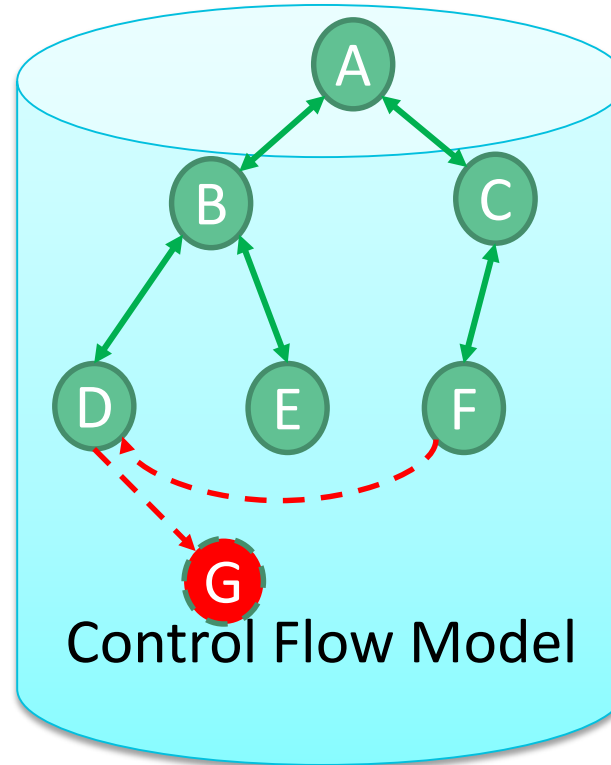
...

# How does it work?

Program execution trace  
during training

A  
B  
D  
B  
E  
B  
A  
C  
F  
C  
A  
...

Control Flow  
Learning



Control Flow  
Checking

Program execution trace  
during Detection

A  
B  
D  
B  
E  
B  
A  
C  
F  
D  
G  
...



# Intel® Processor Trace (Intel® PT)

- A Hardware extension of Intel® Architecture to capture software execution trace with low overhead
- HW generated data packets contain
  - Control flow tracing: Indirect and conditional branches
  - Program and system context information: timestamp, VM context, processor frequency...
  - SW inserted packets: PTWRITE
- Available since Intel 5<sup>th</sup> gen CPUs

# HW Telemetries for Control Flow Tracing

	Intel® Processor Trace (Intel® PT)	Last Branch Record (LBR)	Performance Monitor Interrupt (Trap Frame IP)
Telemetry Density	Dense	Sparse	Sparse
Supported CFI Methods	Coarse & fine grain CFI	Coarse & fine grain CFI	Coarse grain CFI
Support Training	Yes	No	No
Detection Efficacy	Excellent (for both long and short CF attacks)	Good (for long CF attacks)	Good (for long CF attacks)
Suitable Use Cases	Use cases that request high assurance	Use cases that request high performance	Use cases that request high performance

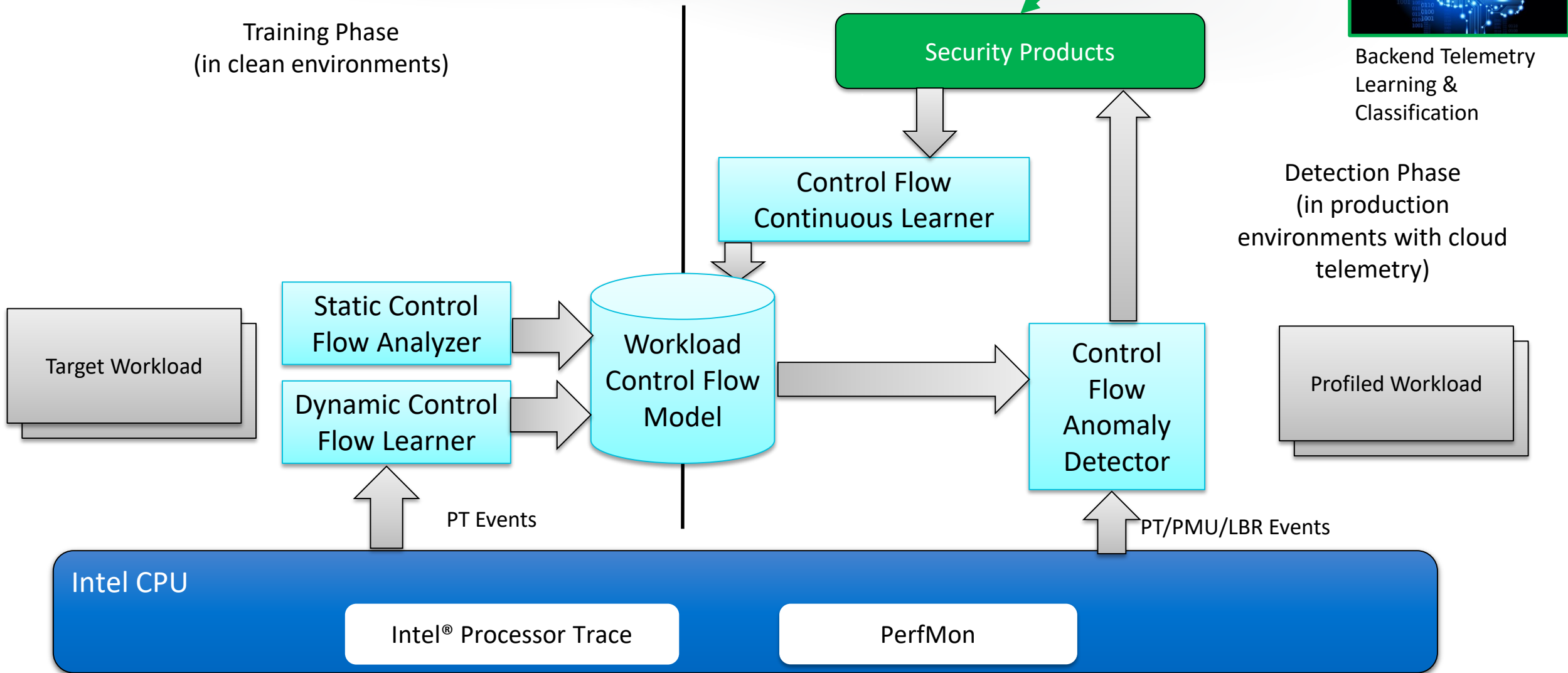
## Recommended configuration for balanced performance and accuracy

- PMI & LBR: always-on baseline monitoring
- PT: on-demand monitoring (based on SW and/or PMI/LBR trigger)



# Training and detection phases

Training Phase  
(in clean environments)



# DEMO



# “Apply” Slide

- Possibilities are endless for threat profiling with HW telemetry.
- 3 month plan
  - Profile a couple threat classes of interest to you or your org
  - Create the control flow models of couple clean workloads
  - Analyze whether your models are working as expected
- 6 month plan
  - Try applying the telemetry identifying and pre-processing techniques to other forms of HW telemetry, e.g. NICs, storage drives, memory.

**RSA**<sup>®</sup>Conference2020

**Questions?**

**RSA**<sup>®</sup>Conference2020

## Additional Reference

# References

- Intel® 64 and IA-32 Architectures Software Developer Manuals  
<https://software.intel.com/en-us/articles/intel-sdm>
  - Volume 3, Chapter 18 – Performance Monitoring
  - Volume 3, Chapter 19 – Performance Monitoring Events
  - Volume 3, Chapter 35 – Intel® Processor Trace
- <https://download.01.org/perfmon> - list of perfmon events for most Intel CPU generations
- <https://github.com/intel-secl/lib-tdt> - open source repo for Intel® Threat Detection Technology telemetry stack.



# **RSA**®Conference2020

## **Key Contributors**

# Key Contributors

- Deepak Kumar Mishra, Intel
- Pablo De Prado, Intel
- Paul Carlson, Intel
- Russ Gorby, Intel
- Shama Mathew, Intel
- Tajunnisha N, Intel