

Hardware is the New Software:
The Next-Gen AntiViruses and how your hardware will
self-secure your system!

Marcus Botacin¹

¹Assistant Professor
Texas A&M University (TAMU), USA
botacin@tamu.edu
@MarcusBotacin

Whoami

Education

- Assistant Professor @ TAMU (Since 2022)
- CS PhD @ UFPR, Brazil (2021)
- CSE/ECE BSc. + CS MSC @ UNICAMP, Brazil (2015, 2017)

Research

- **Malware** at high-level: ML-based detectors.
- **Malware** at mid-level: Sandboxes and tracers.
- **Malware** at low-level: HW-based detectors.

Current Project

- NSF SaTC: Hardware Performance Counters as the next-gen AVs.

Agenda

- 1 Introduction
 - Motivation
- 2 Hardware AVs
 - Core Solutions
- 3 Conclusions
 - Future Directions

Agenda

- 1 Introduction
 - Motivation
- 2 Hardware AVs
 - Core Solutions
- 3 Conclusions
 - Future Directions

The Next-Gen AV must be Dynamic!

Why can't it be static?

We know the limits of static analysis

Limits of Static Analysis for Malware Detection

Andreas Moser, Christopher Kruegel, and Engin Kirda

Secure Systems Lab

Technical University Vienna

{andy, chris, ek}@seclab.tuwien.ac.at

Figure: Source: <https://ieeexplore.ieee.org/document/4413008> (2007)

No Need to Teach New Tricks to Old Malware: Winning an Evasion Challenge with XOR-based Adversarial Samples

Fabrício Ceschin

fjoceschin@inf.ufpr.br

Federal University of Paraná

Curitiba, Paraná, Brazil

Marcus Botacin

mfbotacin@inf.ufpr.br

Federal University of Paraná

Curitiba, Paraná, Brazil

Gabriel Lüders

gl19@inf.ufpr.br

Federal University of Paraná

Curitiba, Paraná, Brazil

Heitor Murilo Gomes

heitor.gomes@waikato.ac.nz

University of Waikato

Hamilton, Waikato, New Zealand

Luiz S. Oliveira

lesoliveira@inf.ufpr.br

Federal University of Paraná

Curitiba, Paraná, Brazil

André Grégio

gregio@inf.ufpr.br

Federal University of Paraná

Curitiba, Paraná, Brazil

Figure: Source: <https://dl.acm.org/doi/10.1145/3433667.3433669> (2021)

This is Zero-Trust. I know!

What is the problem?

What if we trap every memory access?

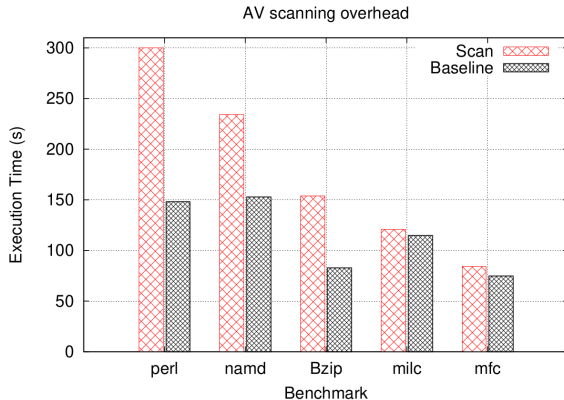


Figure: Source: <https://dl.acm.org/doi/10.1145/3422575.3422775> (2021)

What if we hook every API?

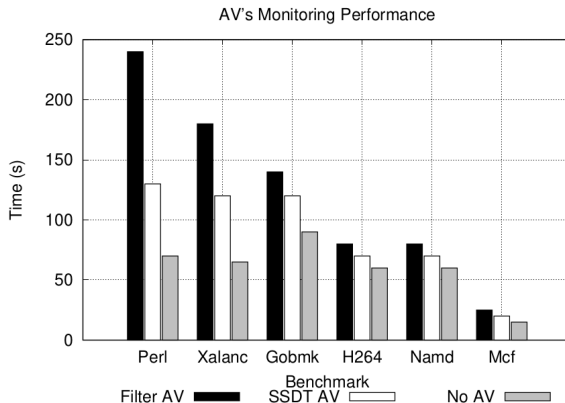


Figure: Source: <https://dl.acm.org/doi/10.1145/3494535> (2022)

We need extra hardware for that!

Agenda

- 1 Introduction
 - Motivation
- 2 Hardware AVs
 - Core Solutions
- 3 Conclusions
 - Future Directions

Weren't Hardware AVs proposed before?

What is wrong with them?

Security Constraints

Memory-efficient signature matching for ClamAV on FPGA

Tran Ngoc Thinh and Tran Trung Hieu
Faculty of Computer Science and Engineering
HCMC University of Technology
Ho Chi Minh City, Vietnam
{tnthinh,hieutt}@cse.hcmut.edu.vn

Hiroshi Ishii and Shigenori Tomiyama
School of Information and Telecommunication Engineering
Tokai University
Tokyo, Japan
ishii164@tokai.ac.jp

Figure: Source: <https://ieeexplore.ieee.org/document/6916730> (2014)

Storage Constraints

MEMORY-Based Hardware Architectures to Detect ClamAV Virus Signatures with Restricted Regular Expression Features

Nga Lam Or, Xing Wang, and Derek Pao, *Member, IEEE*

Figure: Source: <https://ieeexplore.ieee.org/document/7115115> (2016)

Can't we do better than that?
It is 2025...

2-Layer Architecture

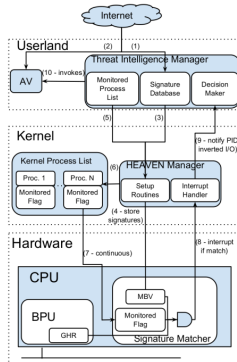


Figure: **Source:**

<https://www.sciencedirect.com/science/article/abs/pii/S0957417422004882>
(2022)

Improving AVs with External Hardware

FPGA AVs for Outlier Detection

What if we add an FPGA to the system?

The AV says: Your Hardware Definitions Were Updated!

Marcus Botacin^{*}, Lucas Galante[†], Fabricio Ceschin^{*}, Paulo C. Santos[‡]

Luigi Carro[‡], Paulo de Geus[†], André Grégio^{*}, Marco A. Z. Alves^{*}

*Federal University of Paraná (UFPR-BR) – {mfbotacin, fjoceschin, gregio, mazalves}@inf.ufpr.br

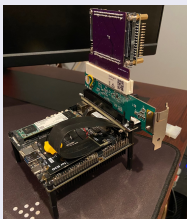
[†]University of Campinas (UNICAMP-BR) – {galante, paulo}@lasca.ic.unicamp.br

[‡]Federal University of Rio Grande do Sul (UFRGS-BR) – {pcssjunior, carro}@inf.ufrgs.br

Figure: Source: <https://ieeexplore.ieee.org/document/9034972> (2019)

What has changed?

Past



Source: <https://github.com/defparam/PCI2Nano-PCB> (2020)

Now

Intel unveils new Xeon chip with integrated FPGA, touts 20x performance boost

By Sebastian Anthony on June 19, 2014 at 1:19 pm [48 Comments](#)

Source: <http://tinyurl.com/y2yabdrp> (2014)

Profiling-Based AV

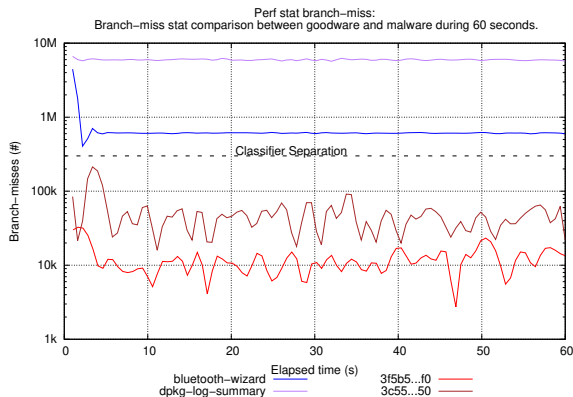


Figure: Malware Classification using low level features.

SVM Classifier

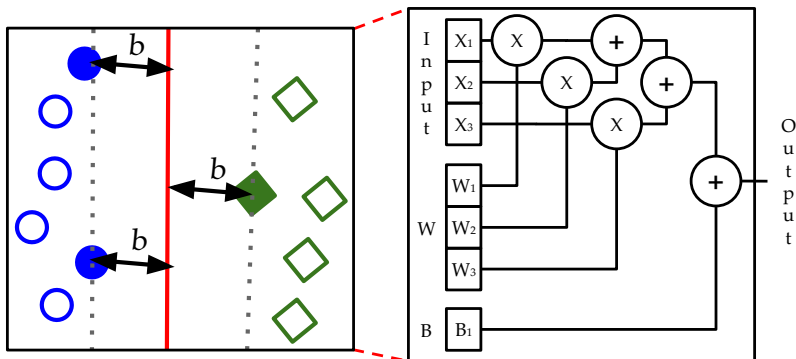


Figure: SVM. A hyperplane with maximum separation between two classes is created and used to predict samples (left). The circuit implemented (right) multiplies the input (x_i) by the learned parameters (w_i) and adds b .

RF Classifier

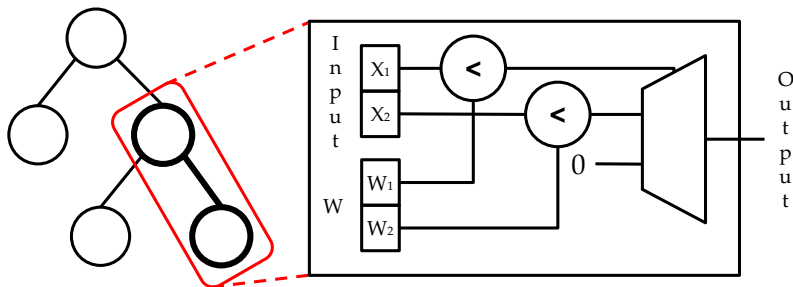


Figure: Random Forest. A single decision tree (from the ensemble) is shown (left) with the corresponding circuit of two nodes (right), with comparators and a MUX deciding the decision path.

MLP Classifier

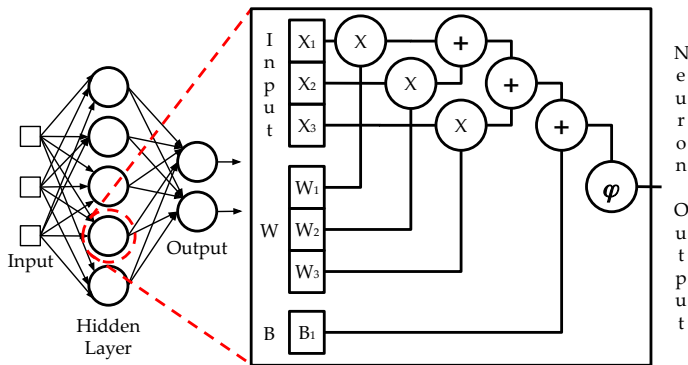


Figure: MLP. A feed-forward neural network composed by multiple neurons (left), which circuit implementation (right) is similar to SVM, but using an activation function to calculate the output.

REHAB Architecture

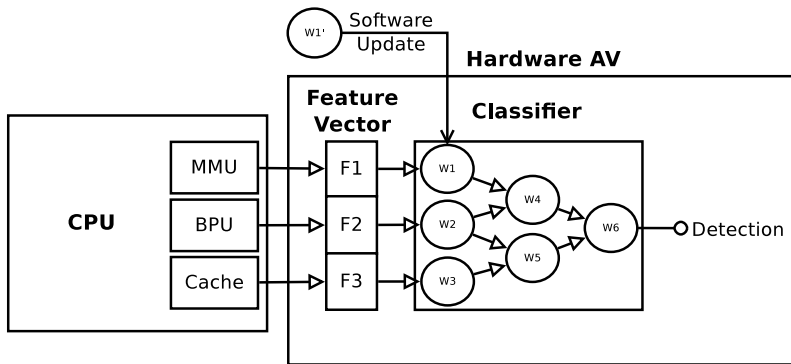


Figure: REHAB Architecture. CPU's HPC data is used as feature for a FPGA-based, reconfigurable ML classifier updatable via software.

REHAB Implementation

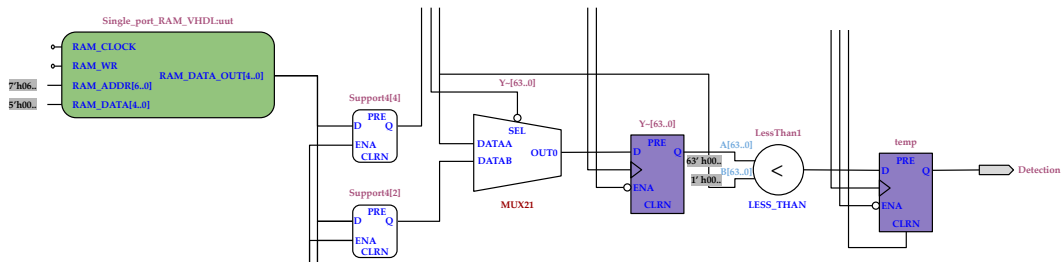


Figure: Excerpt of a ML classifier implemented in FPGA. ML parameters are loaded from an external memory at startup and can be updated by software writes to the external RAM memory.

AV Checks Cost

Table: Execution Speedup per AV check. Hardware Accelerator is essential for overhead elimination.

ML algorithm \rightarrow	SVM	RF	MLP
CPU	$220\mu s$	$270\mu s$	$240\mu s$
FPGA+Comm	$124.5ns$	$111.2ns$	$158.9ns$
Speedup	$1.7k\times$	$2.4k\times$	$1.5k\times$

Improving AVs with Minor CPU Modifications

Detecting Unpacking via Self-Modifying-Code Identification

Hardware Solutions: SMC Detector

Original Paper | Published: 13 February 2020

The self modifying code (SMC)-aware processor (SAP): a security look on architectural impact and support

Marcus Botacin , Marco Zanata & André Grégio

Journal of Computer Virology and Hacking Techniques **16**, 185–196(2020) | [Cite this article](#)

198 Accesses | 3 Altmetric | [Metrics](#)

Figure: **Source:** <https://link.springer.com/article/10.1007/s11416-020-00348-w>
(2020)

UPX: How a packer works?

Table: UPX Sections Memory Mapping. Sections are initially mapped as writable and executable at the same time.

Name	Content	Permissions	Accessed
None	Header	RWE	R
UPX0	Original Code	RWE	RWE
UPX1	Unpacker	RWE	RWE
.rsrc	Resources	RWE	RW

What if we modify the cache?

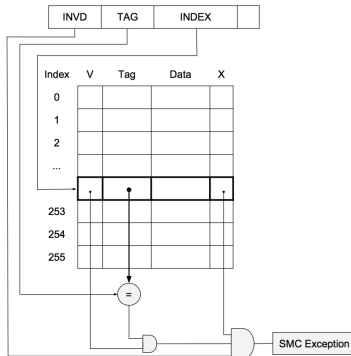


Figure: SMC-Aware Cache. An exception is generated when an invalidation is performed on valid executable cache line.

Then Themida...

```

1  17aa388: 0f ae 7b 74 clflush 0x74(%ebx)
2  198b964: 0f ae 7e bc clflush -0x44(%esi)
3  1cbb375: 0f ae 7a ea clflush -0x16(%edx)
4  1d8ff8e: 0f ae 7e f9 clflush -0x7(%esi)
5  20368e6: 0f ae 7f 18 clflush 0x18(%edi)
6  2349866: 0f ae 3b      clflush (%ebx)

```

Code 1: SMC Code. Forced Cache flush in Themida's loader.

What if we monitor pipeline flushes?

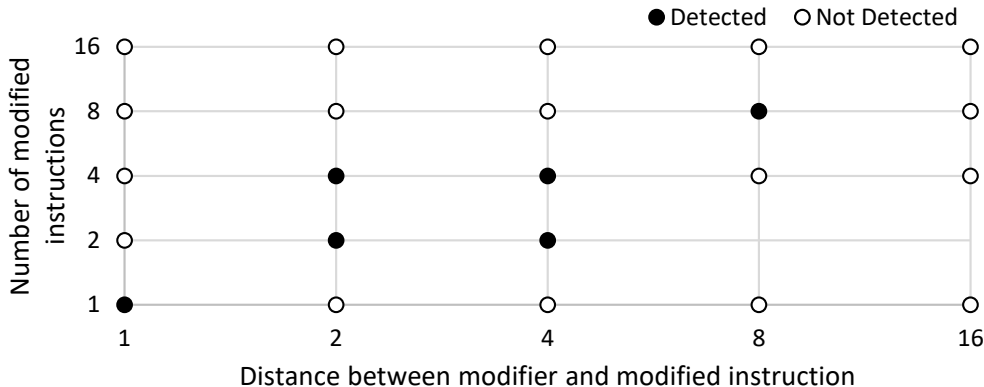


Figure: Effectiveness of event counter as a SMC detector.

What if we monitor the MMU?

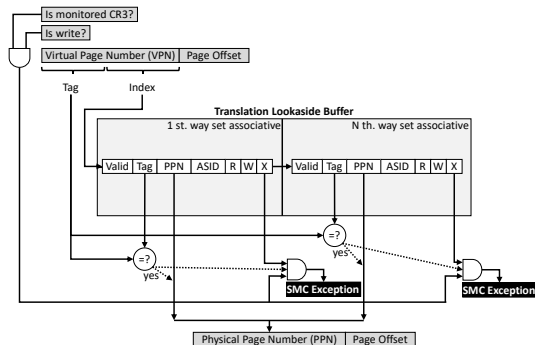


Figure: MMU-based SMC detection mechanism.

What if we monitor the MMU?

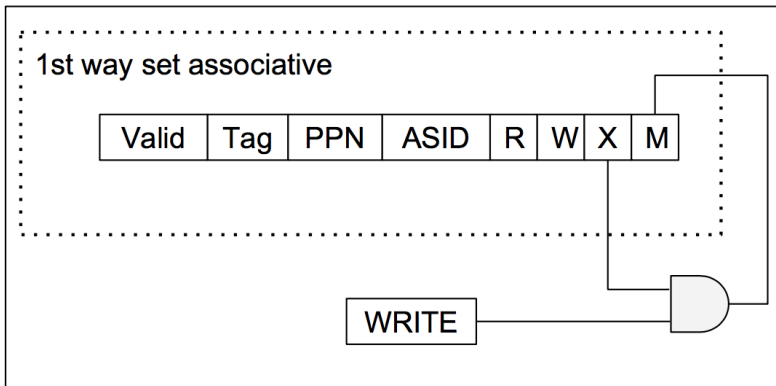


Figure: MMU-based SMC detection mechanism.

Detection Results

Table: SMC Detection. Solutions comparison.

App./ Packer	SMC Type	Cache Change	Cache Flush	Pipeline Flush	MMU	Whitelisted
SPEC	-	<i>X</i>	<i>X</i>	<i>X</i>	<i>X</i>	<i>X</i>
SMC	1	✓	<i>X</i>	<i>X</i>	✓	<i>X</i>
SMC Flush	1	<i>X</i>	✓	<i>X</i>	✓	<i>X</i>
UPX	3	<i>X</i>	<i>X</i>	<i>X</i>	✓	<i>X</i>
Themida	2	<i>X</i>	✓	✓	✓	<i>X</i>
Python	3	<i>X</i>	<i>X</i>	<i>X</i>	✓	✓

Performance Analysis

Table: Estimated overhead of Software-based SMC detectors during page fault trapping on SPEC applications

Benchmark	Penalty	Benchmark	Penalty	Benchmark	Penalty
bzip2	3.47%	mcf	3.76%	wrf	3.91%
namd	1.54%	bwaves	3.73%	perlbench	6.49%
h265ref	4.61%	calculix	3.57%	dealli	3.12%
astar	2.12%	sjeng	2.74%	hmmer	2.62%
gobmk	3.10%	cactusADM	3.70%	libquantum	3.24%
gcc	6.25%	gromacs	4.01%	sphinx3	3.76%
lbm	4.27%	zeusmp	3.48%	povray	4.64%
tonto	4.53%	GemsFDTD	3.48%	xalancbmk	3.85%
gamess	4.05%	leslie3d	3.46%	specrand	3.36%

Improving AVs with Memory Extensions



In-Memory Fileless Memory Scanning

Publication

RESEARCH-ARTICLE

Near-Memory & In-Memory Detection of Fileless Malware



Authors:  [Marcus Botacin](#),  [André Grégio](#),  [Marco Antonio Zanata Alves](#) [Authors Info & Affiliations](#)

Publication: MEMSYS 2020: The International Symposium on Memory Systems • September 2020 • Pages 23–38 • <https://doi.org/10.1145/3422575.3422775>

Figure: **Source:** <https://dl.acm.org/doi/10.1145/3422575.3422775> (2021)

What Is Fileless Malware?

LILY HAY NEWMAN 02.09.17 07:22 PM

Say Hello to the Super-Stealthy Malware That's Going Mainstream

Figure: **Source:** <https://www.wired.com/2017/02/say-heello-super-stealthy-malware-thats-going-mainstream/>

cyberscoop

HEALTHCARE

TECHNOLOGY

FINANCIAL

WATCH

New malware works only in memory, leaves no trace

Figure Source: <https://www.cyberscoop.com/kasperskyy-fileless-malware-memory-attribution-detection/>

A Drawback for Current Security Solutions

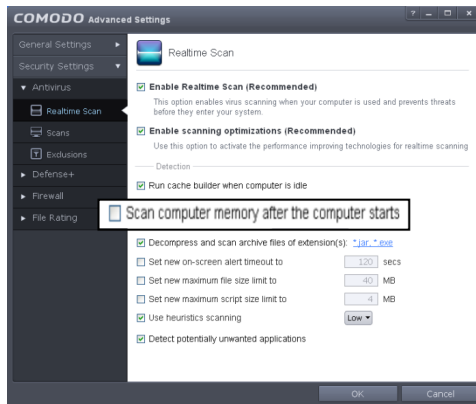


Figure: Default policy is not to scan memory.

Signatures as the Detection Mechanism

```
1  if(IsDebuggerPresent()){
2      evade()
```

Code 2: C code

```
1  mov  eax, [fs:0x30]
2  mov  eax, [eax+0x2]
3  jne  0 <evade>
```

Code 3: ASM code

```
1  64 8b 04 25 30 00 00
2  67 8b 40 02
3  75 e1
```

Code 4: Instructions Bytes

Can't We Rely on Page Faults?

Table: Blocking on Page Faults. The performance impact is greater as more complex is the applied detection routine.

Benchmark	Cycles	PF	5K	10K	20K	30K
perf	187G	1,8M	4,74%	9,48%	18,96%	28,44%
mcf	69G	375K	2,72%	5,45%	10,89%	16,34%
milc	556G	1,2M	1,05%	2,10%	4,21%	6,31%
bzip	244G	170K	0,35%	0,69%	1,38%	2,08%
namd	491G	325K	0,33%	0,66%	1,32%	1,98%

Observing Memory Accesses Patterns

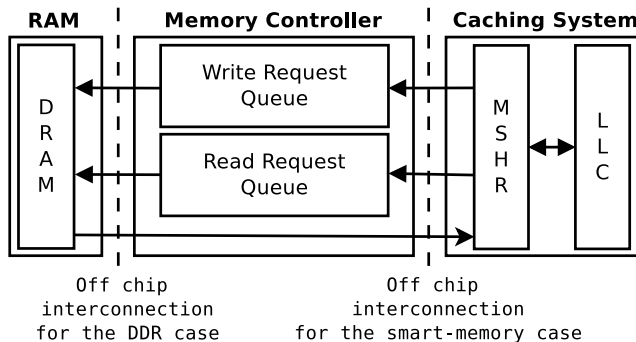


Figure: Write-to-Read window. Read requests originated from the MSHR might overlap other memory-buffered read requests for any address, but must not overlap previous memory-buffered write requests for the same address.

How Much Performance Overhead is Acceptable?

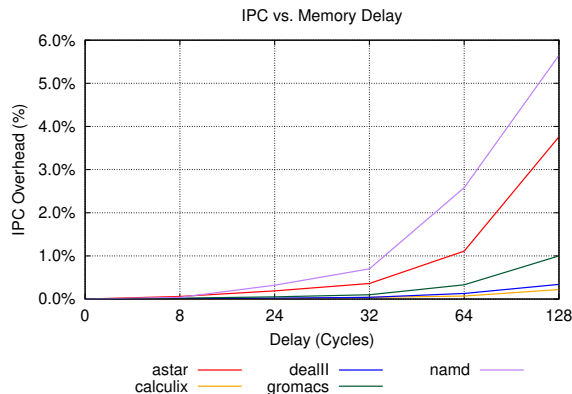


Figure: MINI-ME database overhead. Delays of up to 32 cycles impose less than 1% of IPC overhead.

Key Innovation: Smart-Memories

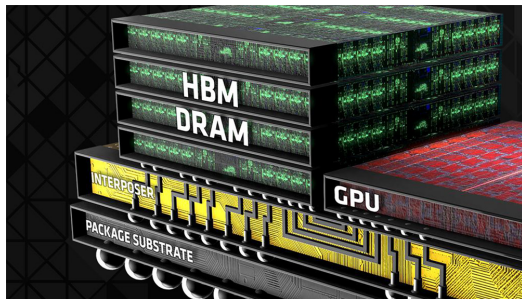


Figure: Source:

<https://pcper.com/2015/05/high-bandwidth-memory-hbm-architecture-and-plans-for-the-future-of-gpus/>

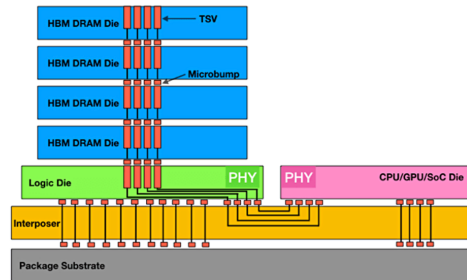


Figure: Source:

<https://semiengineering.com/hbms-future-necessary-but-expensive/>

Malware Identification based on Near- and In-Memory Evaluation (MINI-ME)

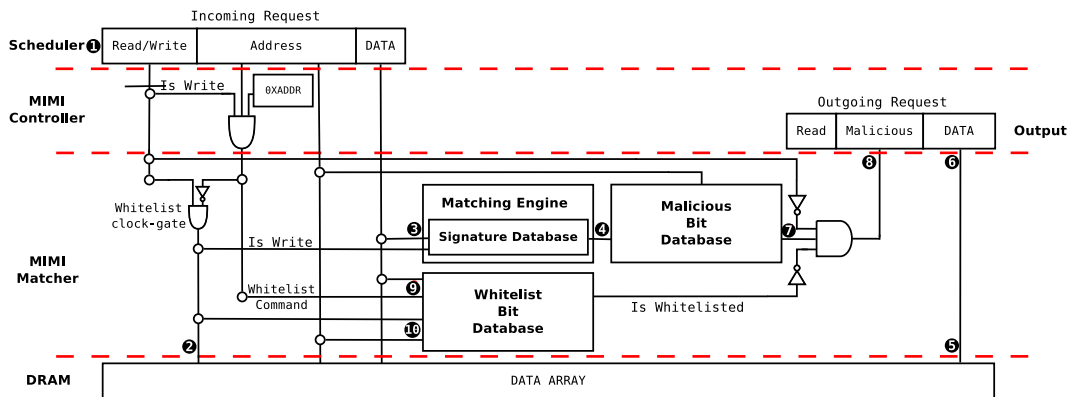


Figure: MINI-ME Architecture. MINI-ME is implemented within the memory controller.

Handling Notifications via Page Faults

```
1 void __do_page_fault(...) {  
2     // Original Code  
3     if (X86_PF_WRITE) ...  
4     if (X86_PF_INSTR) ...  
5     // Added Code  
6     if (X86_MALICIOUS) ...
```

Code 5: Modified PF handler. Malicious bit is set when suspicious pages are mapped.

MINI-ME in Practice

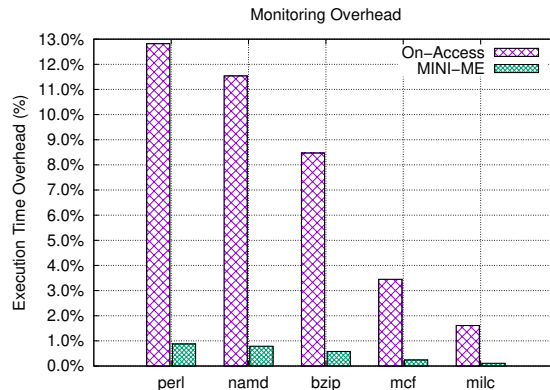


Figure: Monitoring Overhead. MINI-ME imposes a smaller overhead while still checking more pages than an on-access solution.

Improving AVs by Repurposing CPU components

Making the Branch Prediction Unit (BPU) to predict malware execution

Hardware Solutions: BPU Modification



Figure: Source:

<https://www.sciencedirect.com/science/article/abs/pii/S0957417422004882>
(2022)

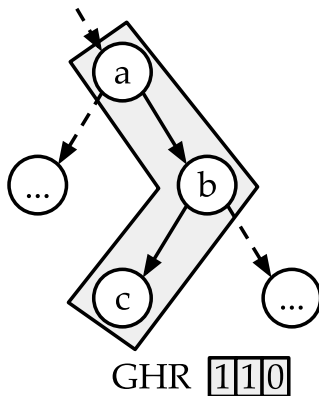
Branch Patterns and Code Patterns

```

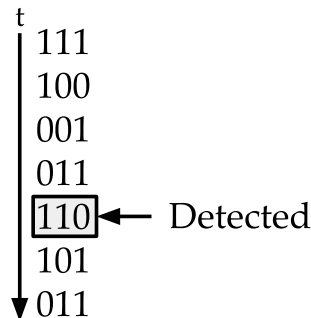
if(a)
    ...
if(b)
    ...
if(c)
    ...
else
    mal(a,b,c)

```

(a) **Code.**



(b) Flow.



(c) **Signature**

Figure: Associating high-level code constructs with their occurrence in the execution flow.

Branch Prediction Background.

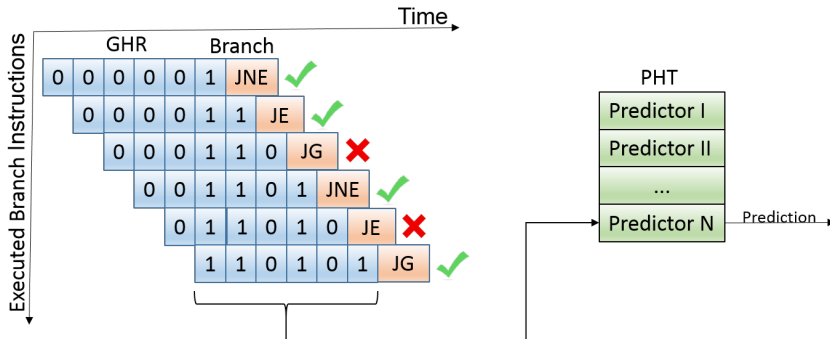


Figure: 2-level branch predictor.

Branch Patterns as Signatures

Table: Signature distribution along code region in the malware samples evaluated. Percentage of good signatures per code region and percentage of malware samples allowing generation of at least one signature for the given code region. A code region [0%-10%] corresponds to the first 10% of the malware trace.

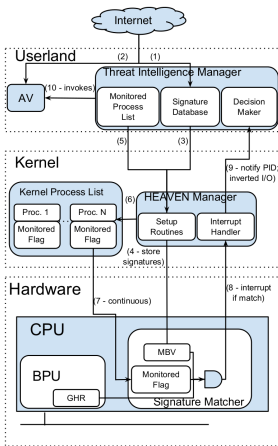
Code region	Signatures	Samples
0%-10%	6%	100%
10%-50%	10%	54%
50%-70%	19%	98%
70%-80%	28%	78%
80%-90%	24%	90%
90%-100%	13%	100%

Branch Patterns as Signatures (3/3)

Table: Malware behaviors associated with HEAVEN produced signatures and the code region in which they are matched (percentage of sample’s execution).

Behavior	Signature prevalence	Code region	Samples
Image Load	18%	0%-10%	100%
Image Launch	45%	0%-10%	100%
File Deletion	81%	80%-90%	100%
Connection	100%	0%-10%	100%
Exfiltration	67%	80%-90%	100%

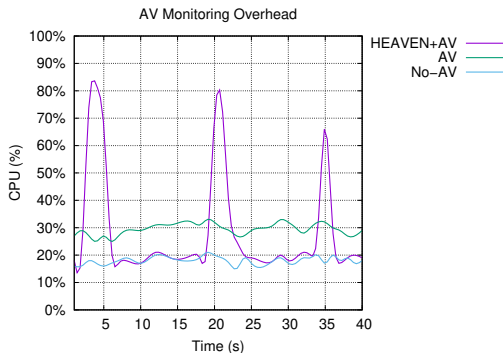
Hardware-Enhanced AntiVirus Engine (HEAVEN)



2-level Architecture

Do not fully replace AVs, but add efficient matching capabilities to them.

Performance Characterization



2-Phase HEAVEN CPU Performance

The inspection phase causes occasional, and quick bursts of CPU usage. The AV operating alone incurs a continuous 10% performance overhead.

Improving AVs with Extra Hardware Resources

A CoProcessor to match YARA rules against API arguments

Hardware Solutions: Real-Time Processor

TERMINATOR: A Secure Coprocessor to Accelerate Real-Time AntiViruses using Inspection Breakpoints

Marcus Botacin, Federal University of Paraná (UFPR-BR)

Francis B. Moreira, Federal University of Rio Grande do Sul (UFRGS-BR)

Philippe O. A. Navaux, Federal University of Rio Grande do Sul (UFRGS-BR)

André Grégio, Federal University of Paraná (UFPR-BR)

Marco A. Z. Alves, Federal University of Paraná (UFPR-BR)

Figure: Source: <https://dl.acm.org/doi/10.1145/3494535> (2022)

YARA rules

```

1 rule IsPacked : PECheck {
2     condition:
3         // MZ signature at offset 0 and
4         uint16(0) == 0x5A4D and
5         // PE signature at offset stored
6         // in MZ header at 0x3C
7         uint32(uint32(0x3C)) == 0x00004550 and
8         math.entropy(0, filesize) >= 7.0
9 }

```

Code 6: YARA rule to detect packed PE files.

Function Hooking

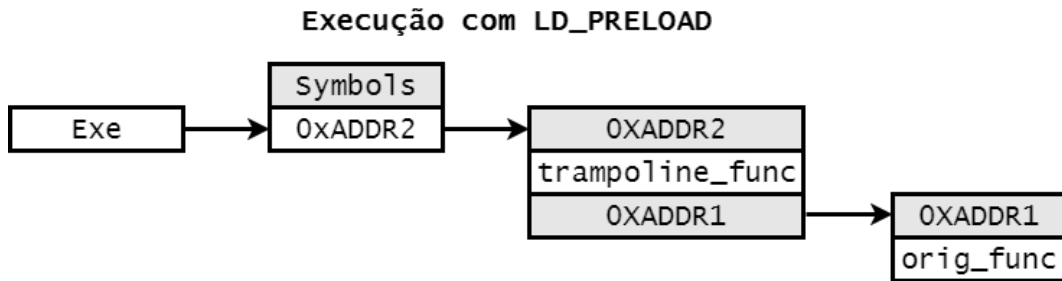


Figure: Function Interposition. A trampoline function is added by the AVs to interpose the original function calls.

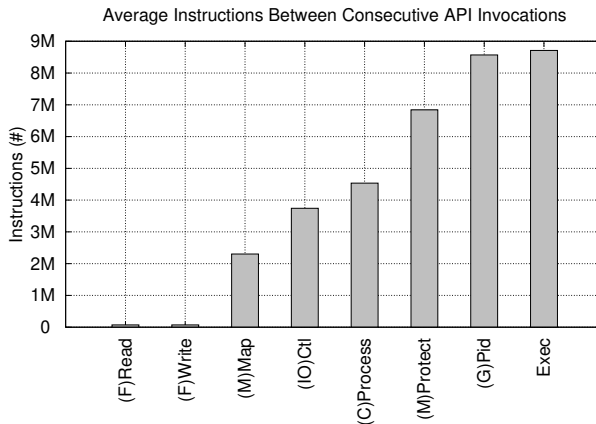


Figure: Number of instructions between consecutive calls of the same function.

Table: Implementation Alternatives. The communication cost diminishes the advantage of using existing processor cores.

Implementation	Ideal			Software		Expected
Cost	Base	Serial	Parallel	Interception	Piped	Piped2
Monitored	✗	✓	✓	✗	✓	✓
API	72K	72K	72K	72K	72K	72K
Interception	0K	0K	0K	34K	34K	0K
Communication	0K	0K	0K	0K	21K	21K
Match	0K	72K	0K (72K)	0K	0K (72K)	0K (72K)
Total	72K	144K	72K	106K	127K	93K
Overhead	0,00%	100,00%	0,00%	47,00%	76,00%	29,00%

CoProcessor Architecture

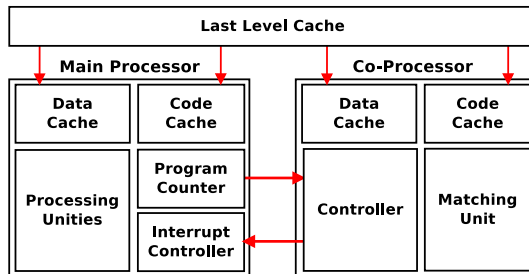


Figure: Co-processor Architecture. The co-processor and CPU share the Last Level Cache but the co-processor is only activated when a program counter registered for monitoration is identified.

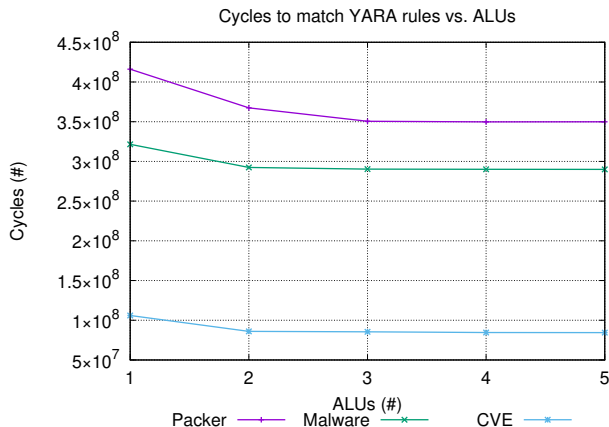


Figure: Cycles to match YARA rules vs. ALU. Adding more ALUs to the coprocessor saves processing cycles.

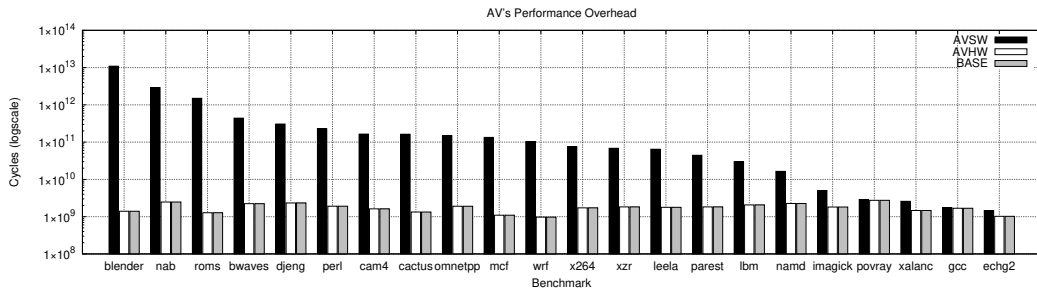


Figure: Performance evaluation when tracking all function calls. Comparison between execution without AV (BASE), execution with software AV, and execution with the proposed coprocessor model.

Agenda

- 1 Introduction
 - Motivation
- 2 Hardware AVs
 - Core Solutions
- 3 Conclusions
 - Future Directions

Future Directions

What to do now?

- Security now has new incentives for demanding extra hardware:
 - Zero-Trust and Static analysis evasion (even with AI).
- AV companies might:
 - Improve AV performance with hardware assistance.
- Hardware manufacturers are understanding that:
 - Performance is not only about clock speed, but also about accelerators.
- Researchers now have:
 - A new research field to explore: Hardware security beyond side-channels.
- AV evaluations should:
 - Start considering AV performance in a new way.

Thanks!
Questions? Comments?
botacin@tamu.edu
@MarcusBotacin