Hardware-accelerated security monitoring

Marcus Botacin

[1]botacin@tamu.edu
marcusbotacin.github.io

Who Am I?

- Assistant Professor (2022) - Texas A&M University (TAMU), USA
  - ACES Program Fellowship
- PhD. in Computer Science (2021) - Federal University of Paraná (UFPR), Brazil
  - Thesis: *"On the Malware Detection Problem: Challenges and new Approaches"*
- MSc. in Computer Science (2017) - University of Campinas (UNICAMP), Brazil
  - Dissertation: *"Hardware-Assisted Malware Analysis"*
- Computer Engineer (2015) - University of Campinas (UNICAMP), Brazil
  - Final Project: *"Malware detection via syscall patterns identification"*

Topics

Bottleneck: Real-time monitoring performance penalty

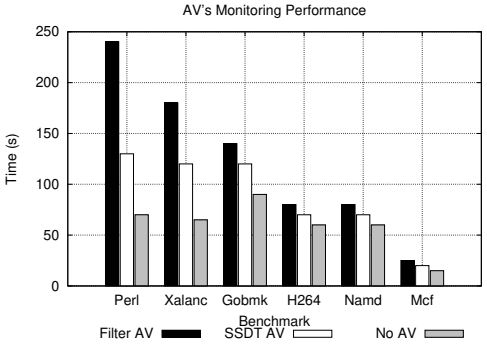


Figure: **AV Monitoring Performance.**

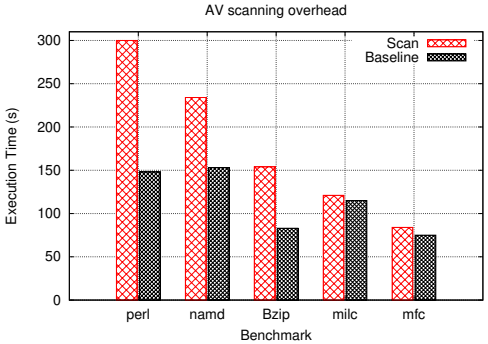
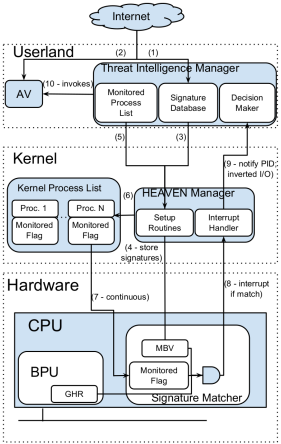


Figure: **In-memory AV scans worst-case and best-case performance penalties.**

Topics

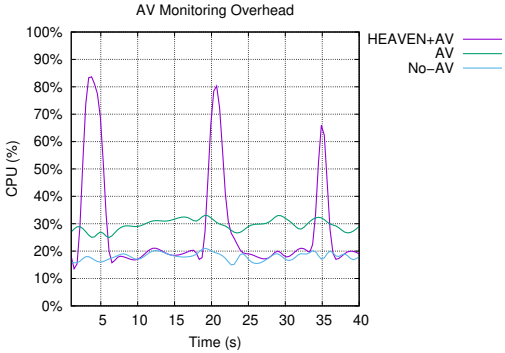
Hardware AV Architecture



2-level Architecture

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

Performance Characterization



2-Phase HEAVEN CPU Performance

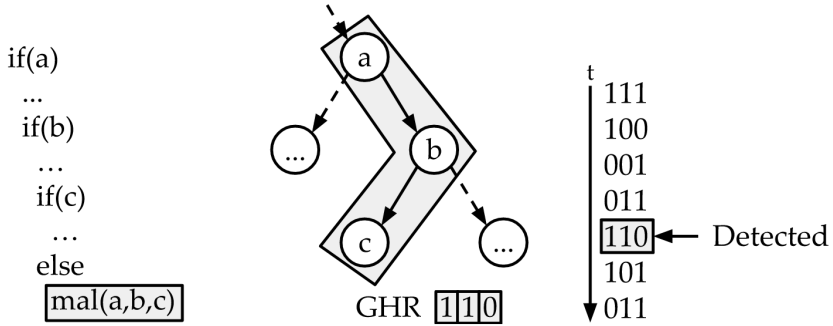
The inspection phase causes occasional, and quick bursts of CPU usage. The AV operating alone incurs a continuous 10% performance overhead.

Topics

Publication

Figure: **Source:** https://www.sciencedirect.com/science/article/abs/pii/S0957417422004882

Branch patterns as signatures



(a) **Code.** (b) **Flow.** (c) **Signature**

Figure 2: **Signature Generation Policy.** Associating high-level code constructs with their occurrence in the execution flow.

Branch patterns as signatures

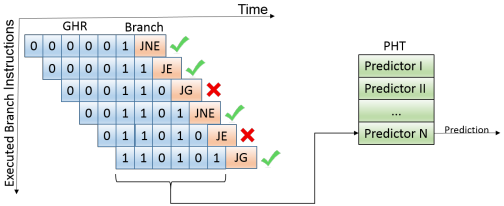


Figure: **Two-level branch predictor.** A sequence window of taken (1) and not-taken (0) branches is stored in the Global History Register (GHR).

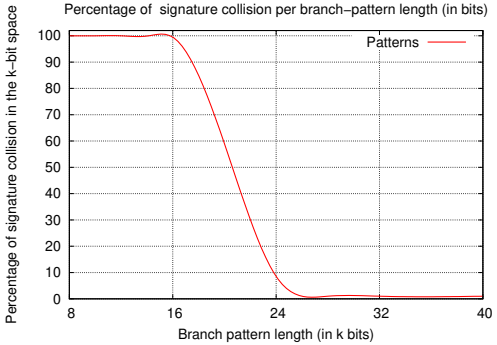
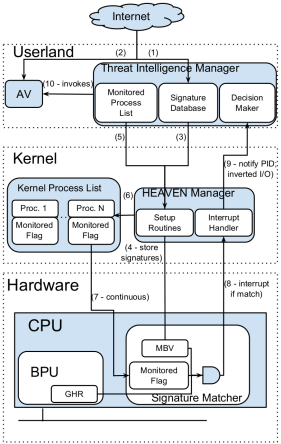


Figure: **Branch patterns coverage.**

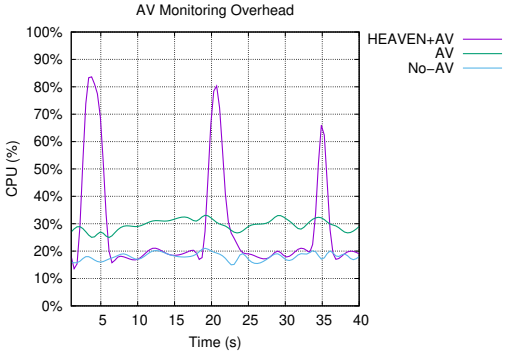
Hardware AV Architecture



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.

Topics

Publication

Figure: **Source:** `https://ieeexplore.ieee.org/document/9034972/`

Profiling-Based AV

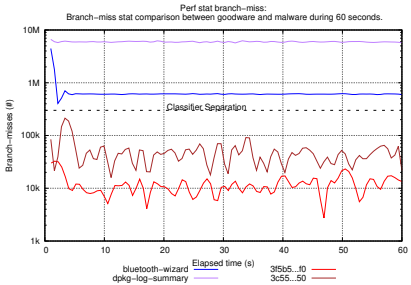


Figure: **Malware Classification using low level features.**

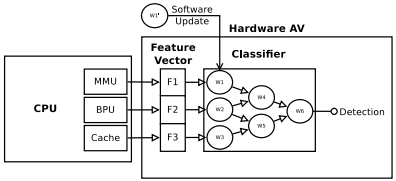


Figure: **REHAB Architecture**. CPU's HPC data is used as feature for a FPGA-based, reconfigurable ML classifier updatable via software.

Classifiers

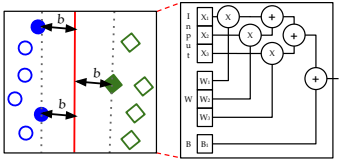


Figure: **SVM**.

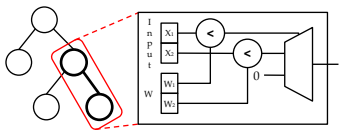


Figure: **Random Forest**.

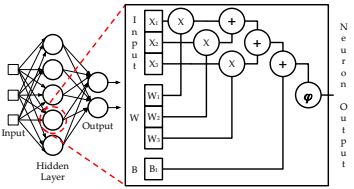


Figure: **MLP**.

AV Checks Cost

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

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

Topics

Publication

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`

Architectural Support

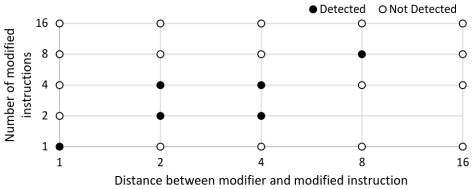


Fig. 2 Effectiveness of event counter as a SMC detector.

Figure: **Pipeline Stalls Detection.**

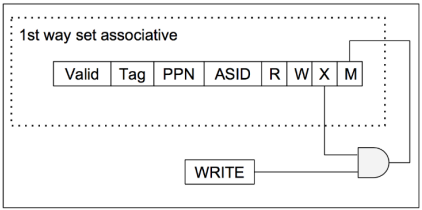


Fig. 4 MMU-based SMC detection mechanism.

Figure: **MMU Modification**.

Page Handling Overhead

```
1 static noinline void __do_page_fault(...)
      {
2 // Original Code
3 if(kprobes())...
4 // Instrumentation Code
5 if(was_executable_page_written()) {
6     if(!is_allowed_process(get_pid()) {
7         // SMC Detected
```

Code 9 SMC detection routines in the Linux kernel. The added verification instructions are executed every page fault.

Figure: **Page Fault Handler.**

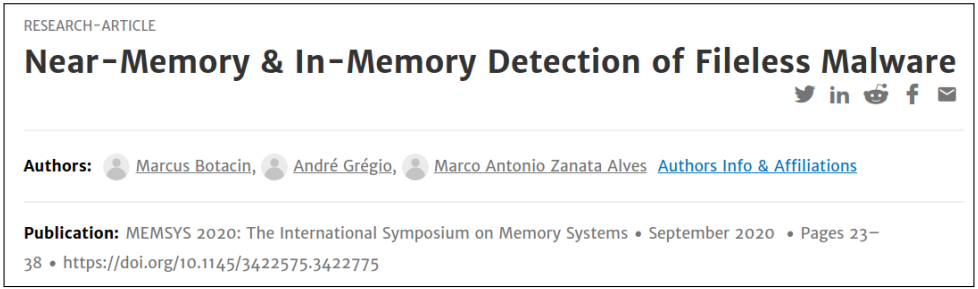
Table 3 Estimated overhead of Software-based SMC detectors during page fault trapping on SPEC applications

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%

Figure: **Performance Penalty**.

Topics

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`

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

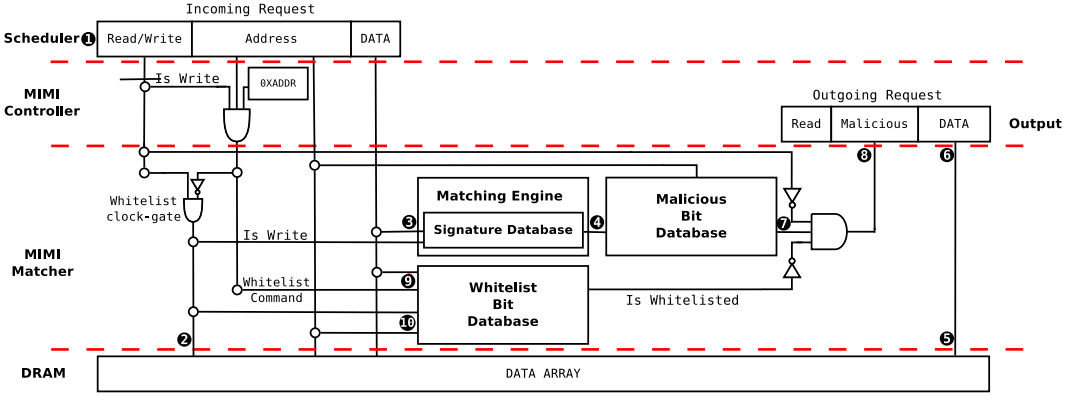
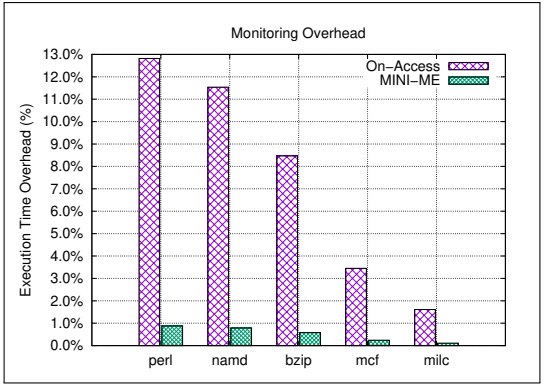


Figure: MINIME Architecture.

Performance Gains

MINIME vs. On-Access AVs

Significant performance gains even in the worst case.



Topics

Publication

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`

Function Arguments Inspection

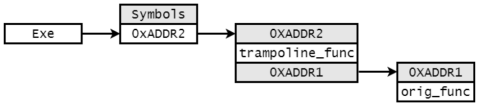


Figure 1: **Function Interposition.** A trampoline function added by AVs to interpose the original function calls.

Figure: **Function Interposition.**

```
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 1: YARA rule to detect packed PE files.

Figure: **Matching Framework**.

Inspection Triggering

```
PcreateProcessNotifyRoutine
    (...){
  pid = GetProcessId();
  libs = EnumProcessModules(
      pid);
  addr = GetModuleAddress(
      libs[target_lib],
      target_function);
  VirtualProtect(addr,
      WRITABLE);
  __intrinsics_set_trap(addr);
  VirtualProtect(addr,
      NOT_WRITABLE|EXECUTABLE)
      ;
}
```

Code 2: Process Creation Callback.

```
1  target_function(...)
2  {
3      entry_checkpoint(
4          empty_slot()
5      );
6      first_task();
7      internal_calls();
8  ...
9      exit_checkpoint();
10     return;
11 }
```

Code 3: Target Function Code.

Figure: **Inspection Breakpoint.**

Parallel Execution Constraints

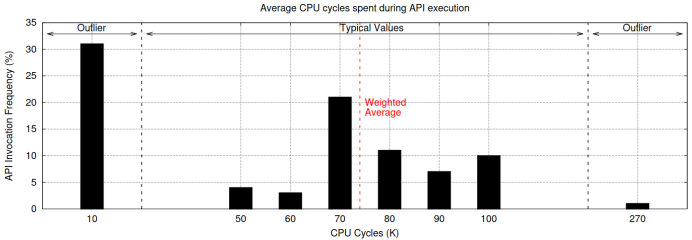


Figure 10: **Weighted Average.** Extreme values are unlikely to be monitored and thus were discarded.

Figure: **Scanning Cycles Boundary.**

Performance Penalty Reduction

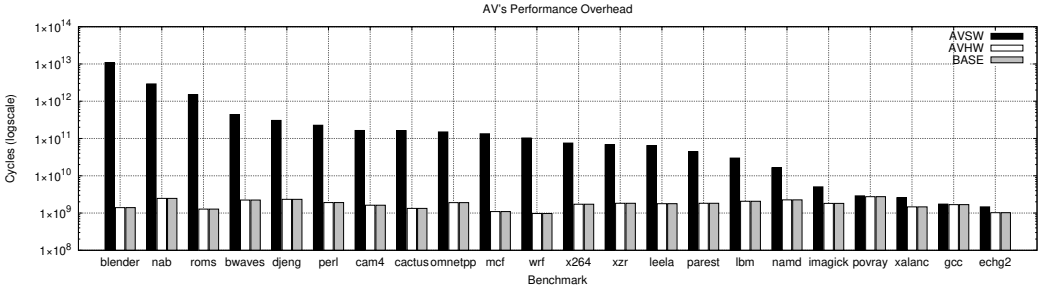


Figure: **Performance evaluation when tracking all function calls.** Comparison between execution without AV (BASE), execution with software AV, and execution with the proposed coprocessor model.

Topics

Summary

Malware Detection

- Huge performance penalties.
- Increasing performance increases detection.

Academic Contributions

- Branch patterns to replace byte-based signatures.
- FPGAs to classify HPCs in runtime.
- SMC-aware processor to detect packers.
- Instrumented memory controller to detect fileless malware.
- CPU coprocessors for real-time syscall checking.

Thanks!

Questions? Comments?

@MarcusBotacin
botacin@tamu.edu
marcusbotacin.github.io