Near-memory & In-Memory Detection of Fileless Malware

Marcus Botacin[1]

[1]Texas A&M University (TAMU)
botacin@tamu.edu

SBSEG 2023

Agenda

Agenda

1. Introduction
2. Proposed Solution
3. Evaluation
4. Conclusions

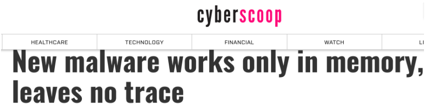
0x0. What is the most concerning type of malware these days?

Fileless malware on the news


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-hello-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/kaspersky-fileless-malware-memory-attribution-detection/`

0x1. How do fileless malware work?

Fileless malware infection chain

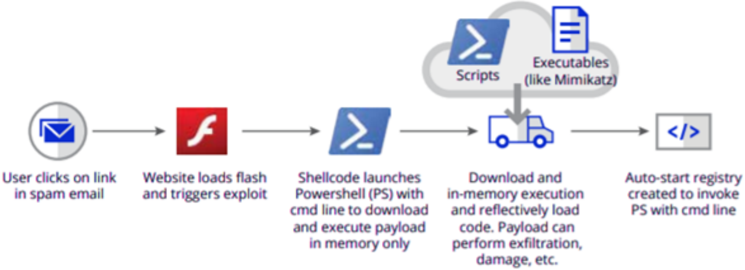


Figure: **Source:** `https://www.trellix.com/en-us/security-awareness/ransomware/what-is-fileless-malware.html`

0x2. How hard is to go fileless?

Fileless malware generation tooks

Figure: **Source:** `https://github.com/nnsee/fileless-elf-exec`

0x3. Is this a real threat?

Fileless malware in the wild

```python
import ctypes, os, base64, zlib
l = ctypes.CDLL(None)
s = l.syscall
c = base64.b64decode(b'eNrsvXlcVOX3OH4HGBZFZ3CLzI
e = zlib.decompress(c)
f = s(319, '', 1)
os.write(f, e)
p = '/proc/self/fd/%d' % f
os.execle(p, 'smd', {})
```

Figure: **Source:** `https://www.wiz.io/blog/pyloose-first-python-based-fileless-attack-on-cloud-workloads/`

0x4. Are current AVs ready for that?

A Drawback for Current Security Solutions

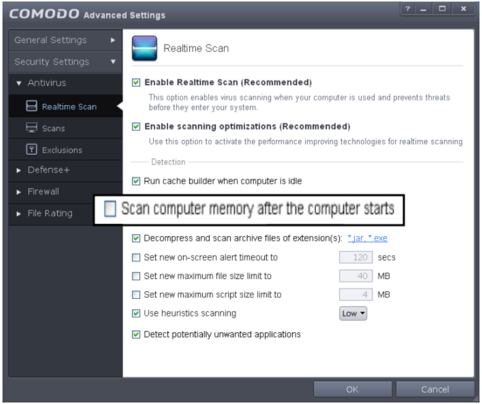


Figure: **Default policy is not to scan memory.**

0x5. Why not to scan all the time?

The Cost of Scanning Memory

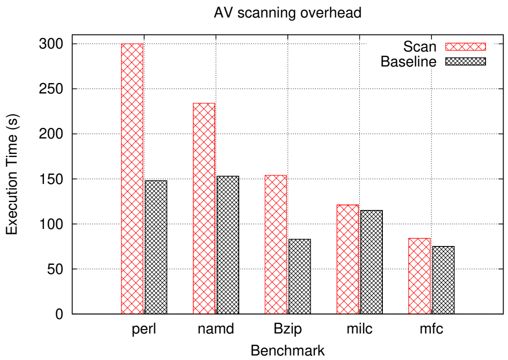


Figure: **In-memory AV scans worst-case and best-case performance penalties.**

0x6. Where does this overhead come from?

0x6.1 How do we detect malware?

Publication

Computers & Security
Available online 12 October 2021, 102500
In Press, Journal Pre-proof

AntiViruses under the Microscope: A Hands-On Perspective

Marcus Botacin [a], Felipe Duarte Domingues [b], Fabrício Ceschin [a], Raphael Machnicki [a], Marco Antonio Zanata Alves [a], Paulo Lício de Geus [b], André Grégio [a]

Figure: **Source:** `https://www.sciencedirect.com/science/article/pii/S0167404821003242`

AV Detection Mechanisms

Table: **Deobfuscation Functions.** Not all techniques are applied to entire payloads.

Technique	XOR			BASE64			RC4			Embedding/Carving		
Mode	**Sig.**	**RT**	**OD**	**Sig.**	**RT**	**OD**	**Sig.**	**RT**	**OD**	**Sign.**	**RT**	**OD**
Avast		✗	✗	✓	✗	✓		✗	✗		✗	✗
MalwareBytes		✗	✗	✓	✗	✗		✗	✗		✗	✗
VIPRE		✗	✗	✓	✗	✗		✗	✗		✗	✗
Kaspersky		✗	✗	✓	✓	✓		✗	✗		✗	✗
TrendMicro		✗	✗	✓	✗	✗		✗	✗		✗	✗

Signatures as the Detection Mechanism

```c
if(IsDebuggerPresent()){
  evade()
```

Code 1: C code

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

Code 2: ASM code

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

Code 3: Instructions Bytes

0x6.1.1 Are signatures still widely-used?

Signature Prevalence

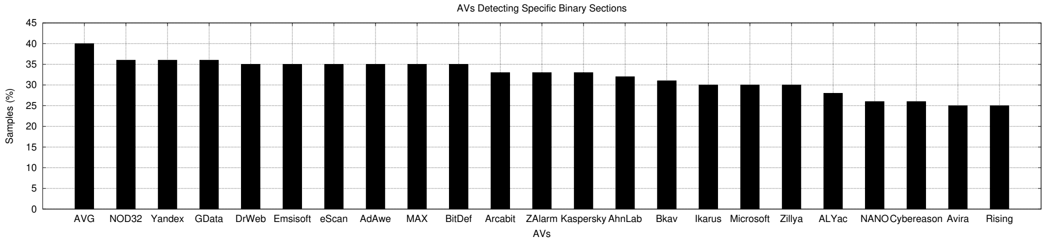


Figure: **Signature Prevalence.** Around a third of the AV's detections are based on specific section's contents.

0x6. Where does this overhead come from?

Memory Dumping Techniques

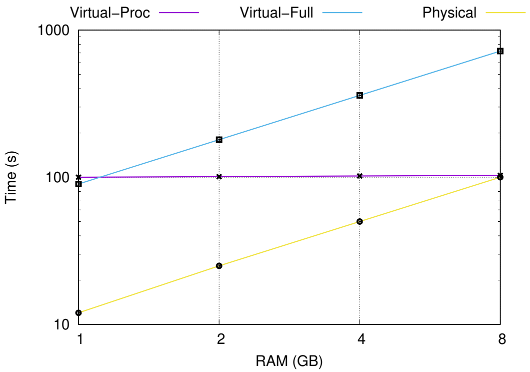


Figure: **Memory dump time for distinct software-based techniques and memory sizes.**

0x7. Is there a way to eliminate this performance cost?

Publication

0

HEAVEN: a Hardware-Enhanced AntiVirus ENgine to accelerate real-time, signature-based malware detection

Marcus Botacin, Federal University of Paraná (UFPR-BR)
Marco A. Z. Alves, Federal University of Paraná (UFPR-BR)
Daniela Oliveira, University of Florida (UFL-US)
André Grégio, Federal University of Paraná (UFPR-BR)

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

Understanding Malware Detection Tasks

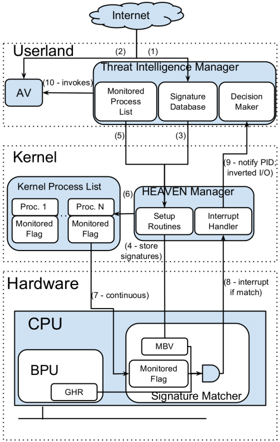
Monitoring

- **You need to know:** When to inspect.

Classifying

- **You need to know:** What to inspect.

Hardware-Enhanced AntiVirus Engine (HEAVEN)



2-level Architecture

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

0x8. Why not using existing hardware?

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%

Agenda

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: **Link:** `https://dl.acm.org/doi/10.1145/3422575.3422775`

0x9. How does the memory work?

What can we explore?

Observing Memory Accesses Patterns

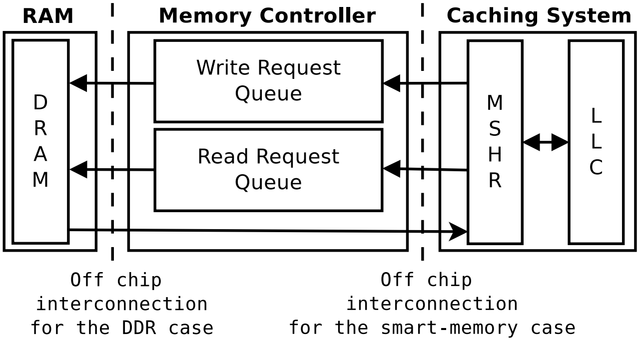


Figure: **Write-to-Read window.** Read requests originated from the MSHR might overlap other memory-buffered read requests for any address, but must not overlap previous memory-buffered write requests for the same address.

0xA. How does the hardware detector look like?

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

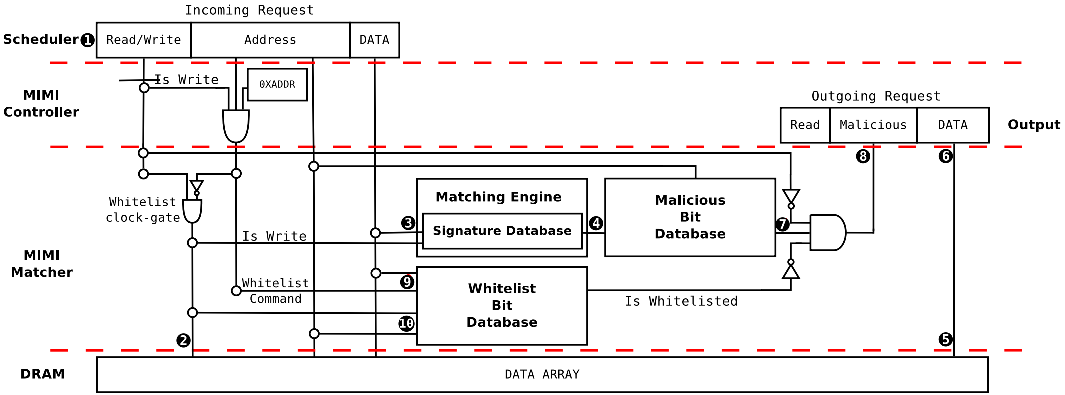


Figure: **MINI-ME Architecture.** MINI-ME is implemented within the memory controller.

0xB. How does the software know about hardware detections?

Handling Notifications via Page Faults

```
void __do_page_fault(...) {
    // Original Code
    if (X86_PF_WRITE) ...
    if (X86_PF_INSTR) ...
    // Added Code
    if (X86_MALICIOUS) ...
```

Code 4: **Modified PF handler.** Malicious bit is set when suspicious pages are mapped.

Agenda

0xC. How many CPU cycles can we delay?

How Much Performance Overhead is Acceptable?

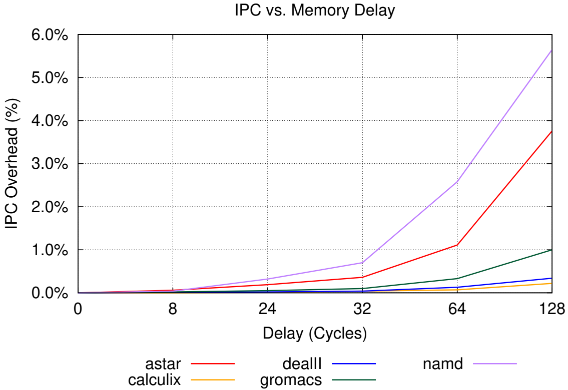


Figure: **MINI-ME database overhead.** Delays of up 32 cycles impose less than 1% of IPC overhead.

0xD. What signature size should we use?

Signature Size Definition

Table: **Signature Generation**. Signatures (%) detected as false positives for each signature size and memory dump size.

		Memory Size			
		1 GB	2 GB	4 GB	8 GB
Signature Size	8 B	8.65%	9.92%	10.18%	11.45%
	16 B	3.06%	3.32%	3.32%	3.32%
	32 B	0.00%	0.00%	0.00%	0.00%
	64 B	0.00%	0.00%	0.00%	0.00%

0xE. Which storage type should we use?

Matching Mechanism Definition

Table: **Matching Techniques.** FP rates for multiple signature sizes and techniques.

		Signature size			
		8 B	**16 B**	**32 B**	**64 B**
Match. Tech.	**Dir. Mapped Table**	8.33%	3.15%	0.00%	0.00%
	Signature Tree	8.33%	3.15%	0.00%	0.00%
	Bloom Filter	8.41%	3.47%	0.00%	0.00%

0xF. What scan policy should we use?

Matching Policies Definition

Table: **Scan Policies.** FP rate for multiple signature sizes and policies.

		Signature size			
		8 B	**16 B**	**32 B**	**64 B**
Scan Policy	**Whole Memory**	8.33%	3.15%	0.00%	0.00%
	Mapped Pages	0.06%	0.01%	0.00%	0.00%
	Whitelist	0.00%	0.00%	0.00%	0.00%
	Code-Only	0.01%	0.00%	0.00%	0.00%

0xF+1 (OOB). Is it better than a software-based, on-access AV?

MINI-ME in Practice

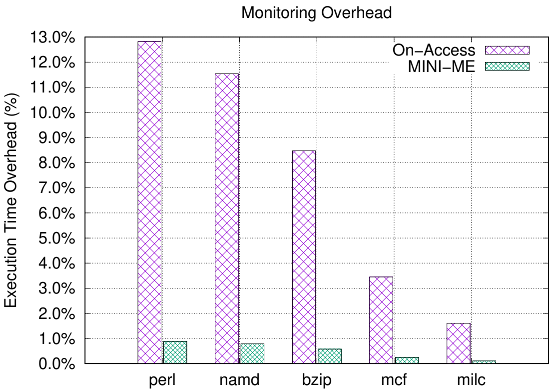


Figure: **Monitoring Overhead.** MINI-ME imposes a smaller overhead while still checking more pages than an on-access solution.

Agenda

Conclusions

Challenges & Lessons

- Fileless malware is a growing hard-to-detect class of threats.
- Traditional AntiViruses (AVs) impose significant performance overhead to perform memory scans.
- In-memory and Near-memory AVs helps reducing AV's performance overheads.
- The more complex the matching mechanism, the greater the performance overhead.
- MINI-ME as platform for future developments.

Questions & Comments.

Contact

- **botacin@tamu.edu**
- **@MarcusBotacin**

Additional Material

- `https://github.com/marcusbotacin/In.Memory`
- `https://marcusbotacin.github.io/`

Looking Ahead

- I'm looking for PhD students!