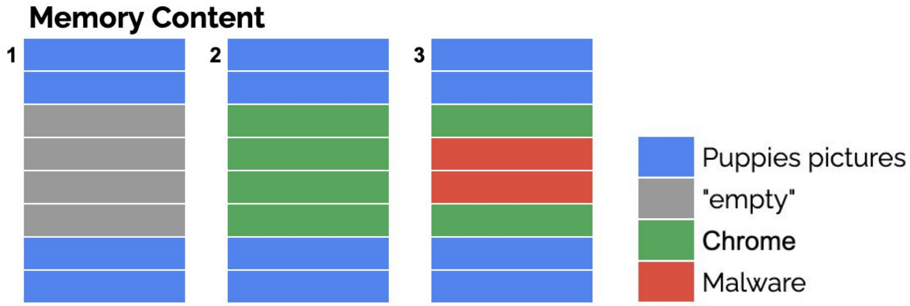
Detecting fileless malware using Endpoint Detection and Response tools

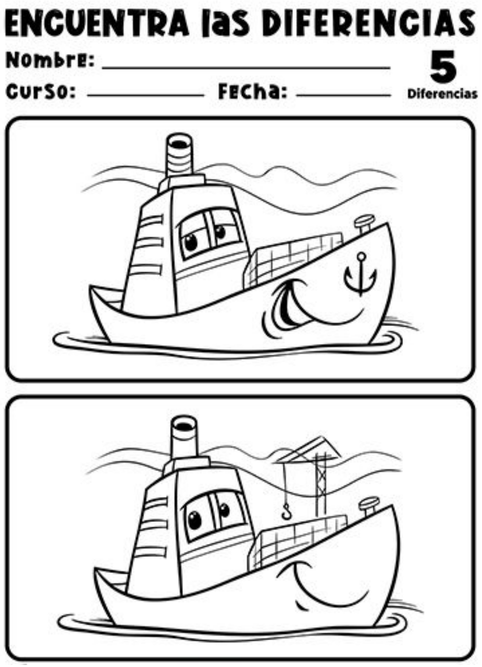
Lautaro Lecumberry
Supervised by Michael Denzel and Nicolás Wolovick

What is the idea?

- Executing processes: HDD -> RAM -> CPU
- We compare the code stored on disk of the programs with the code loaded in RAM of the programs.
- Hypothesis: if the content changed, we were hacked.



What have we done?



A malware detection technique: Mem2Disk

- Based on the comparison of the code between memory and disk.
- **Objective: to detect fileless malware.**

Hypothesis: if the content changed, we were hacked.

What do we want to detect?
Fileless malware

- Fileless malware: This is a type of malware that does not use executables as its main resource to carry out the attack.
  - It uses legitimate and trusted processes and tools to attack and then hide.

What do we want to detect?

Process injection and process hollowing

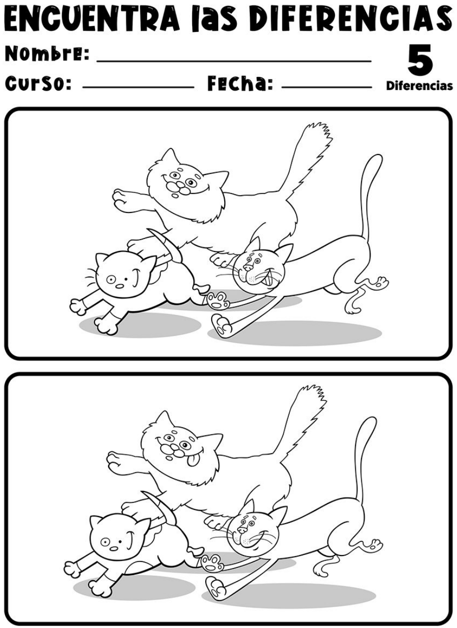
Process injection is a technique that consists of injecting your own code into the memory space of another process.

Process hollowing is a sub-technique of process injection: code injection is done in a "controlled" way.

1. Create a new process.
2. Allocate executable segment in virtual memory.
3. Write the new segment.
4. (optional) unmap the original code section.
5. Restart the process.

Note: The signatures do not change because the content on disk does not change.

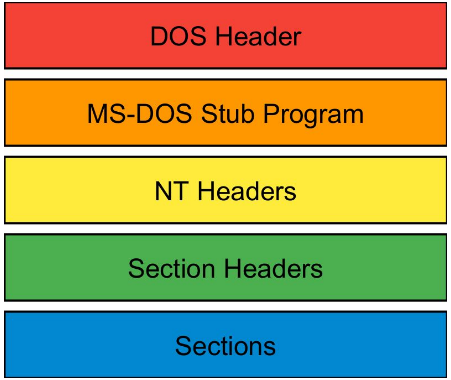
What have we done?



A malware detection technique: Mem2Disk

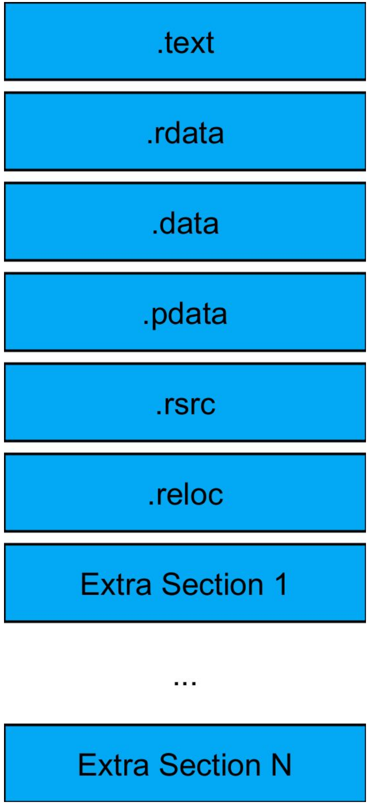
- **Based on the comparison of the code between memory and disk.**
- Objective: to detect fileless malware.

What is compared?
Portable Executable file format

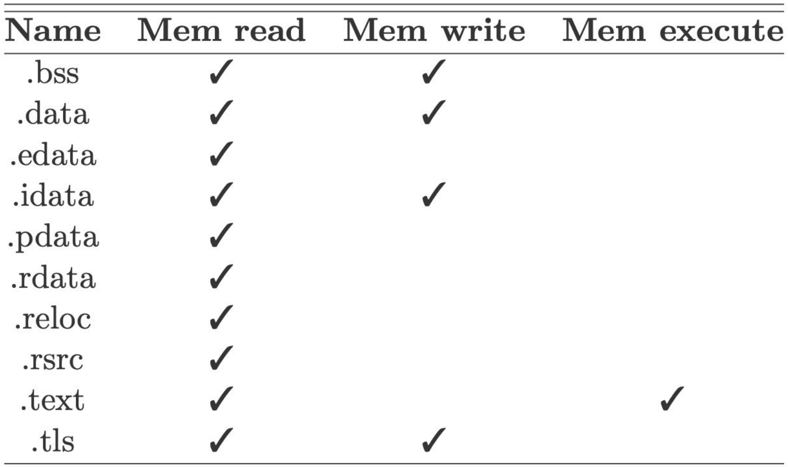


- The PE file format is a type of files which specifies the structure of executable files and object files in the entire Windows family.
- Common extensions *.exe, .dll*

What is compared?
PE FF in memory



- Each of these sections are present in memory with different types of permissions (x, r, w).



Name	Mem read	Mem write	Mem execute
.bss	✓	✓	
.data	✓	✓	
.edata	✓		
.idata	✓	✓	
.pdata	✓		
.rdata	✓		
.reloc	✓		
.rsrc	✓		
.text	✓		✓
.tls	✓	✓	

What technique do we use?
Digital Forensics

What is it?

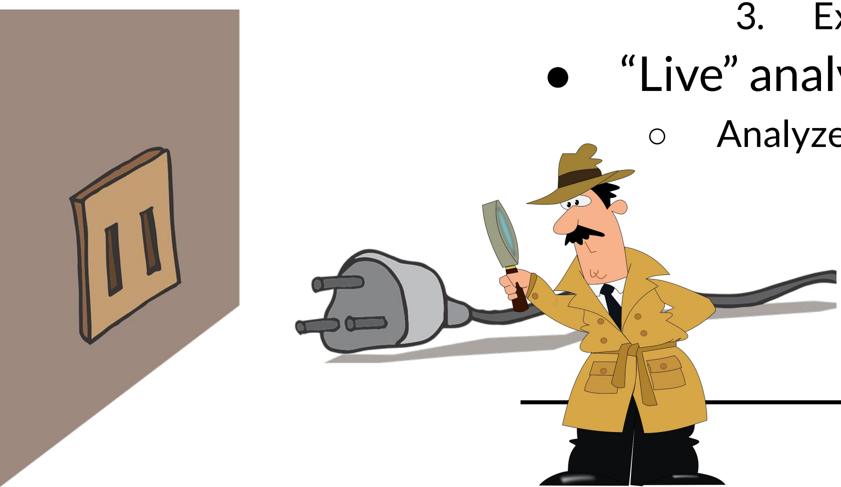
- Forensics: scientific methods of solving crimes by examining objects or substances related to them[1].
- Digital Forensics: Determine what has happened to the computer.

[1]: Cambridge Dictionary

What technique do we use? Digital Forensics

How is it done?

- “Dead” analysis.
    1. Turn off the computer.
    2. Create an image of the storage device.
    3. Examine the image.
- “Live” analysis
    - Analyze the computer while it is turned on.



What technique do we use?
Memory forensics

What is it?

Forensic analysis of RAM.

How is it done?

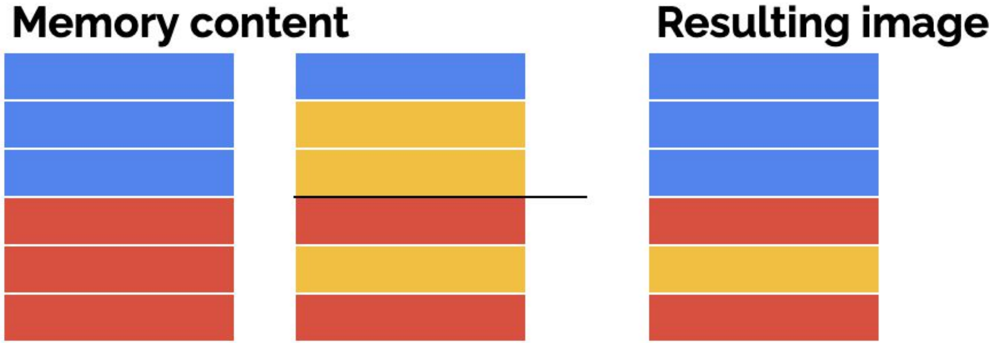
a. Create a physical memory image.
b. Analyze it (frameworks: volatility, rekall, ...).

What technique do we use?
Live memory forensics

- It is the combination of live forensics and memory forensics.
- As it happens live, it is not reproducible.
- Depends on the OS.

Live memory forensics: Pros

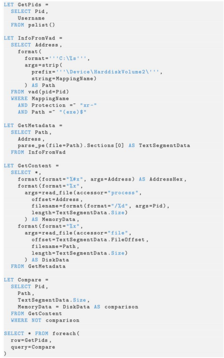
- By not turning off the computer, you can access:
  - Swapped pages.
  - Demand pages.
- It avoids page smearing



Final result 1b Mem2Disk: Velociraptor artifact!

Steps:

1. Get PID of all the processes.
2. Get address of all the executable sections on memory.
3. Get path of all the executable sections on disk.
4. Access PE file headers on disk.
5. Get the content from memory and disk.
6. Compare the contents with each other.
7. Do it for all the processes.



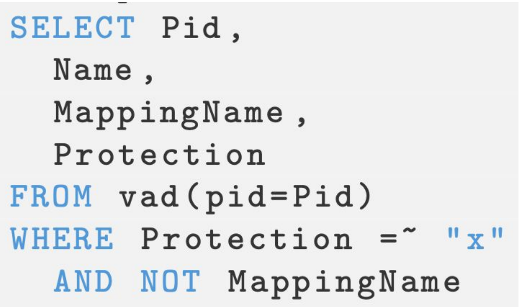
```
LET GetPids =
  SELECT Pid,
    Username
  FROM pslist()

LET InfoFromVad =
  SELECT Address,
    format(
      format='''C:\%s''',
      args=strip(
        prefix='''\Device\HarddiskVolume2\''',
        string=MappingName)
      ) AS Path
  FROM vad(pid=Pid)
  WHERE MappingName
    AND Protection =~ "xr-"
    AND Path =~ "(exe)$"

LET GetMetadata =
  SELECT Path,
    Address,
    parse_pe(file=Path).Sections[0] AS TextSegmentData
  FROM InfoFromVad

LET GetContent =
  SELECT *,
    format(format="%#x", args=Address) AS AddressHex,
    format(format="%x",
      args=read_file(accessor="process",
        offset=Address,
        filename=format(format="/%d", args=Pid),
        length=TextSegmentData.Size)
      ) AS MemoryData,
    format(format="%x",
      args=read_file(accessor="file",
        offset=TextSegmentData.FileOffset,
        filename=Path,
        length=TextSegmentData.Size)
      ) AS DiskData
  FROM GetMetadata

LET Compare =
  SELECT Pid,
    Path,
    TextSegmentData.Size,
    MemoryData = DiskData AS comparison
  FROM GetContent
  WHERE NOT comparison

SELECT * FROM foreach(
  row=GetPids,
  query=Compare
)
```

https://github.com/lautarolecumberry/DetectingFilelessMalware

Final result 1b ExtraX: Velociraptor artifact!

Steps:

1. Get the PID of all processes.
2. Get all the sections with executable permissions and no mapping name.
3. Do it for all the processes.



```
SELECT Pid,
  Name,
  MappingName,
  Protection
FROM vad(pid=Pid)
WHERE Protection =~ "x"
  AND NOT MappingName
```

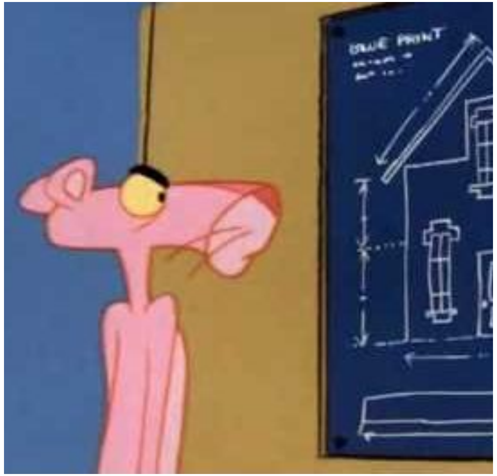
https://github.com/lautarolecumberry/DetectingFilelessMalware

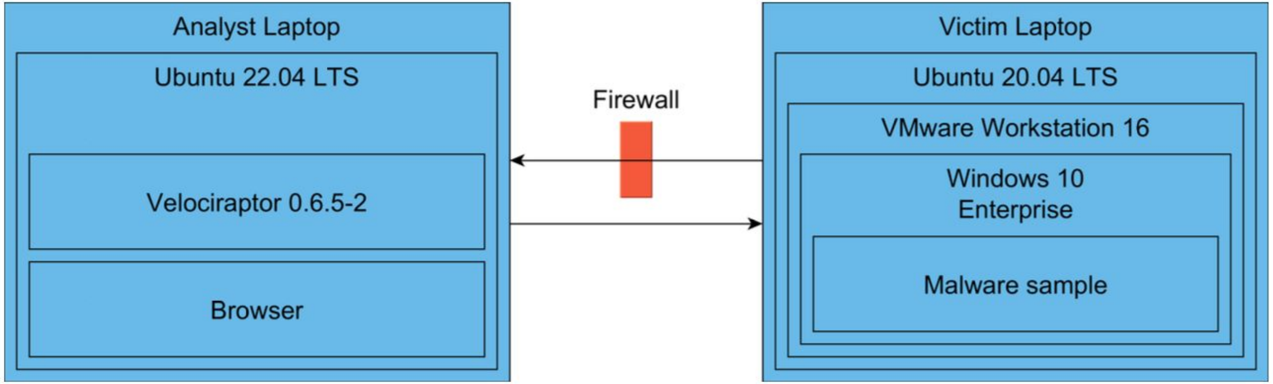
Let's get down to the fun stuff!

What was tested?

- We tested whether this technique detects real malware.
- With mostly publicly available malware.

How was it tested?
Architecture





Analyst Laptop

Ubuntu 22.04 LTS

Velociraptor 0.6.5-2

Browser

Firewall

Victim Laptop

Ubuntu 20.04 LTS

VMware Workstation 16

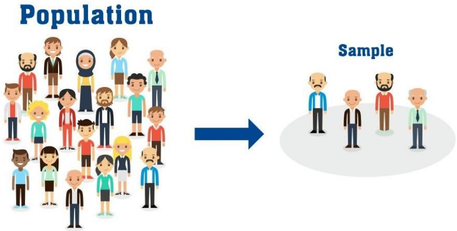
Windows 10 Enterprise

Malware sample

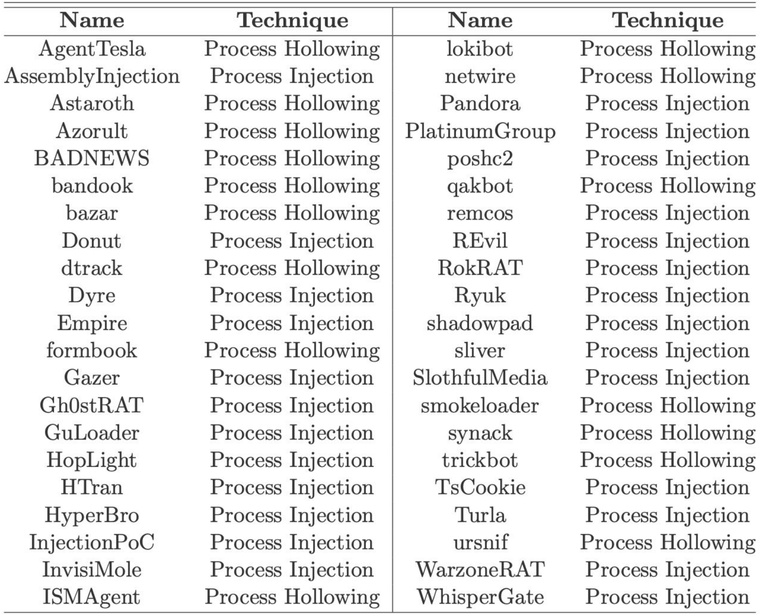
Figure 3.1: Architecture for malware testing.

How was it tested?
Samples

- I obtained 79 samples from 41 different families (downloaded from bazaar, thezoo, vx-underground).
- In addition to the public samples, the team's pentesters gave me two other samples.



How was it tested?
Downloaded families



Name	Technique	Name	Technique
AgentTesla	Process Hollowing	lokibot	Process Hollowing
AssemblyInjection	Process Injection	netwire	Process Hollowing
Astaroth	Process Hollowing	Pandora	Process Injection
Azorult	Process Hollowing	PlatinumGroup	Process Injection
BADNEWS	Process Hollowing	poshc2	Process Injection
bandook	Process Hollowing	qakbot	Process Hollowing
bazar	Process Hollowing	remcos	Process Injection
Donut	Process Injection	REvil	Process Injection
dtrack	Process Hollowing	RokRAT	Process Injection
Dyre	Process Injection	Ryuk	Process Injection
Empire	Process Injection	shadowpad	Process Injection
formbook	Process Hollowing	sliver	Process Injection
Gazer	Process Injection	SlothfulMedia	Process Injection
Gh0stRAT	Process Injection	smokeloader	Process Hollowing
GuLoader	Process Injection	synack	Process Hollowing
HopLight	Process Injection	trickbot	Process Hollowing
HTran	Process Injection	TsCookie	Process Injection
HyperBro	Process Injection	Turla	Process Injection
InjectionPoC	Process Injection	ursnif	Process Hollowing
InvisiMole	Process Injection	WarzoneRAT	Process Injection
ISMAgent	Process Hollowing	WhisperGate	Process Injection

How was it tested?
Detonating malware!

Steps to follow:

1. Victim: retrieve the snapshot.

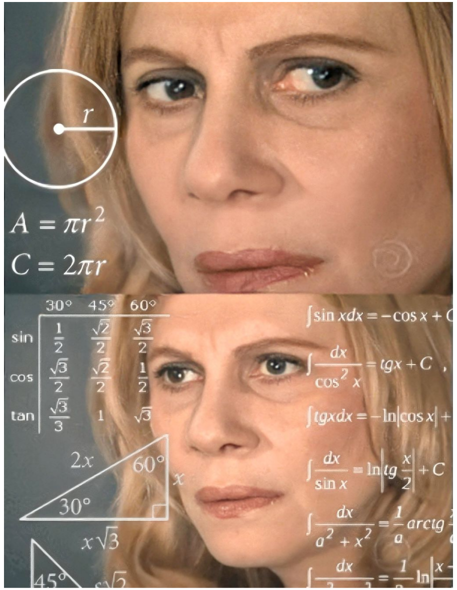
2. Victim: move the malware from the host to the guest virtual machine.

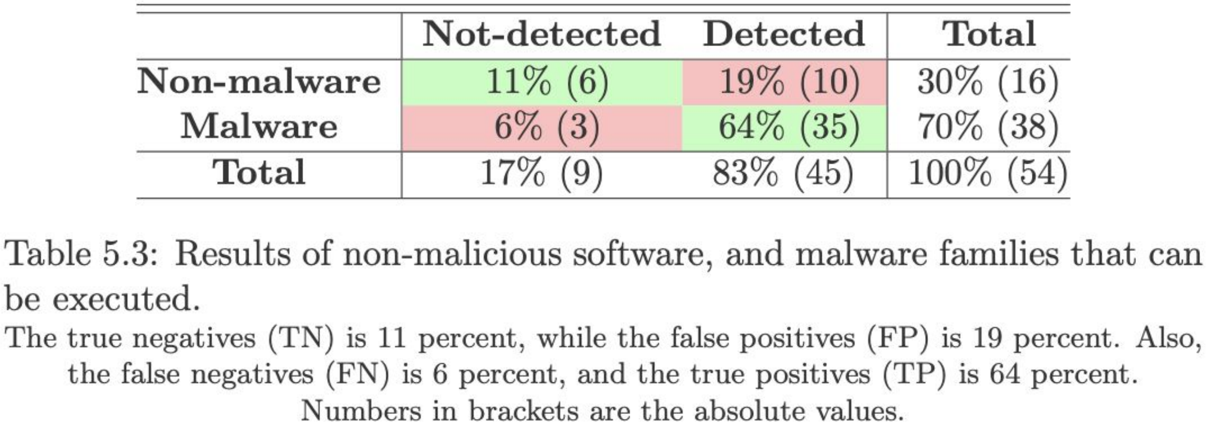
3. Analyst: collect data with Velociraptor to learn the state of the computer before the malware is detonated.

4. Victim: detonate the malware.

5. Analyst: see if Velociraptor queries detected the malware.

Testing phase results



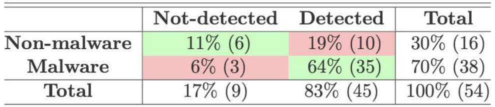


	Not-detected	Detected	Total
Non-malware	11% (6)	19% (10)	30% (16)
Malware	6% (3)	64% (35)	70% (38)
Total	17% (9)	83% (45)	100% (54)

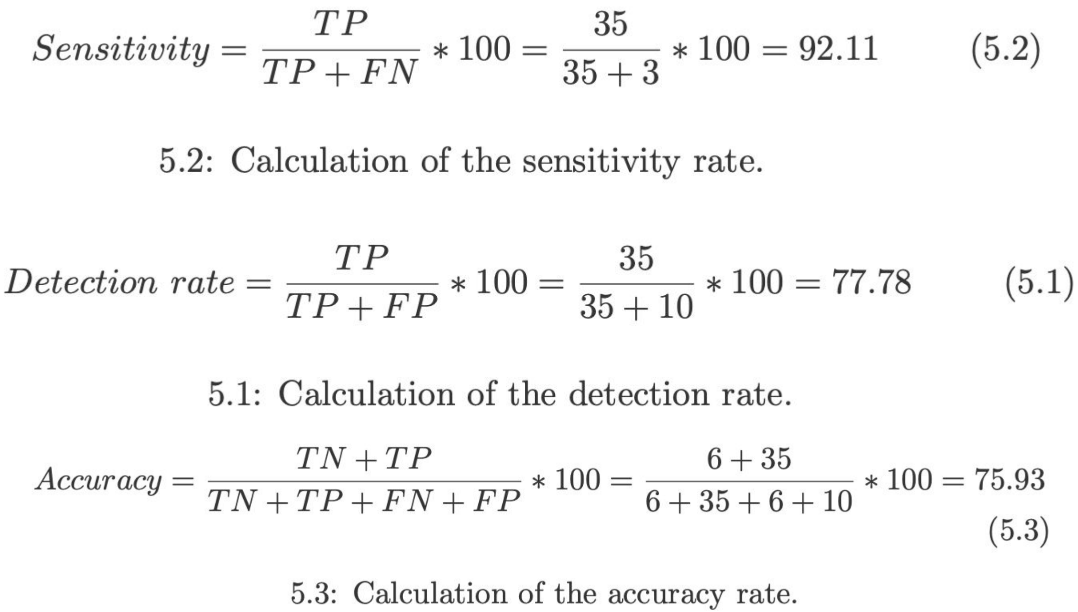
Table 5.3: Results of non-malicious software, and malware families that can be executed.

The true negatives (TN) is 11 percent, while the false positives (FP) is 19 percent. Also, the false negatives (FN) is 6 percent, and the true positives (TP) is 64 percent. Numbers in brackets are the absolute values.

Testing phase results:
Rates



	Not-detected	Detected	Total
Non-malware	11% (6)	19% (10)	30% (16)
Malware	6% (3)	64% (35)	70% (38)
Total	17% (9)	83% (45)	100% (54)



$$Sensitivity = \frac{TP}{TP + FN} * 100 = \frac{35}{35+3} * 100 = 92.11 \tag{5.2}$$

5.2: Calculation of the sensitivity rate.

$$Detection\ rate = \frac{TP}{TP + FP} * 100 = \frac{35}{35+10} * 100 = 77.78 \tag{5.1}$$

5.1: Calculation of the detection rate.

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} * 100 = \frac{6+35}{6+35+6+10} * 100 = 75.93 \tag{5.3}$$

5.3: Calculation of the accuracy rate.

False negatives

- With a false negatives of 6 percent, we believe it is not a priority to focus on reducing it.
- This percentage be maintained while the other limitations are addressed.

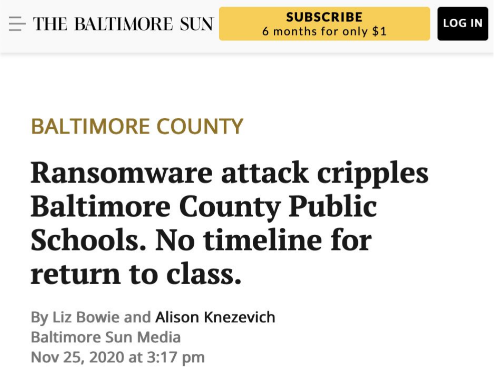
True negatives

Timing issues:

- The attack is happening but it is too fast to be detected.
- The processes are already terminated -> it is no longer possible to access to the process memory.

True positives:
Ryuk Family

- It was necessary to suspend the process to see its condition.
- When I access the memory, it is possible to detect a modification in the code segment of icacls.exe, one of the processes being created by the Ryuk process.



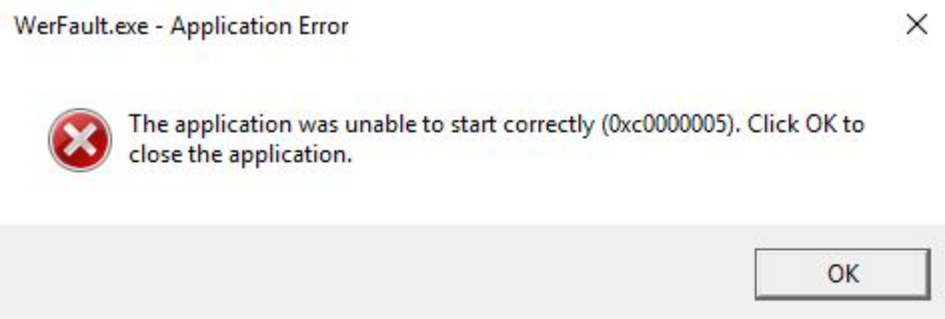
BALTIMORE COUNTY

Ransomware attack cripples Baltimore County Public Schools. No timeline for return to class.

By Liz Bowie and Alison Knezevich
Baltimore Sun Media
Nov 25, 2020 at 3:17 pm

True positives: WhisperGate and remcos

- WhisperGate and remcos do process injection in *WerFault.exe*. This process is the one that triggers the warning sign in Windows.
- Mem2Disk detects it.



WerFault.exe - Application Error

The application was unable to start correctly (0xc0000005). Click OK to close the application.

True positives:
Analysis

Section of the memory code after injecting code:

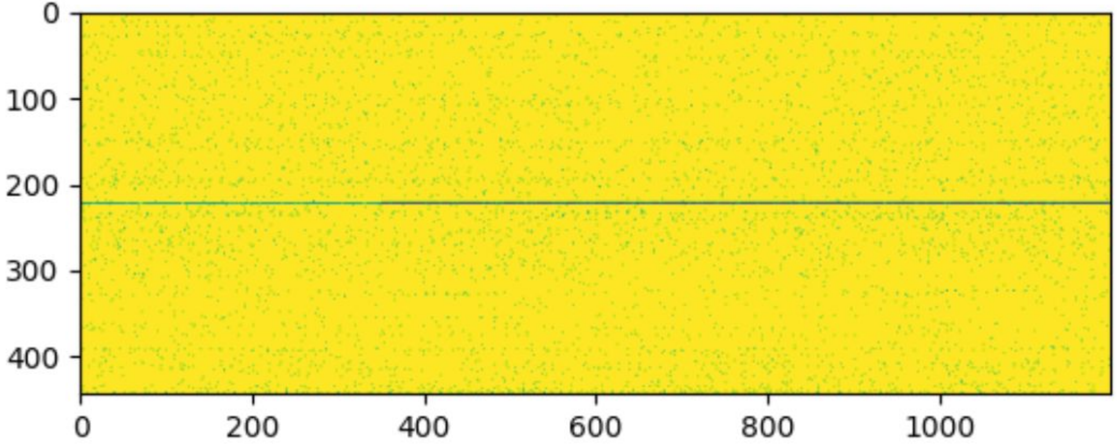


Figure 5.2: Injected malware code segment bitmap.

False positives:
Legitimate software

Bytes keep changing:

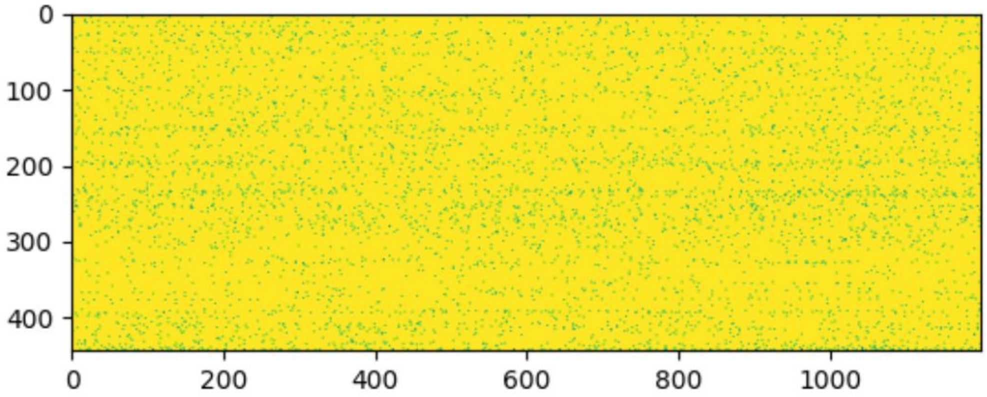
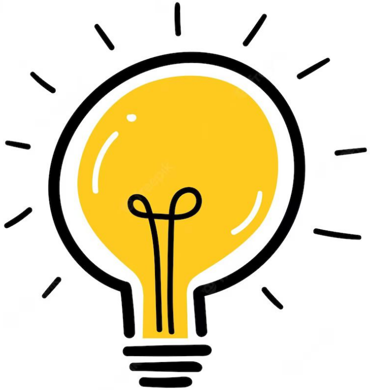
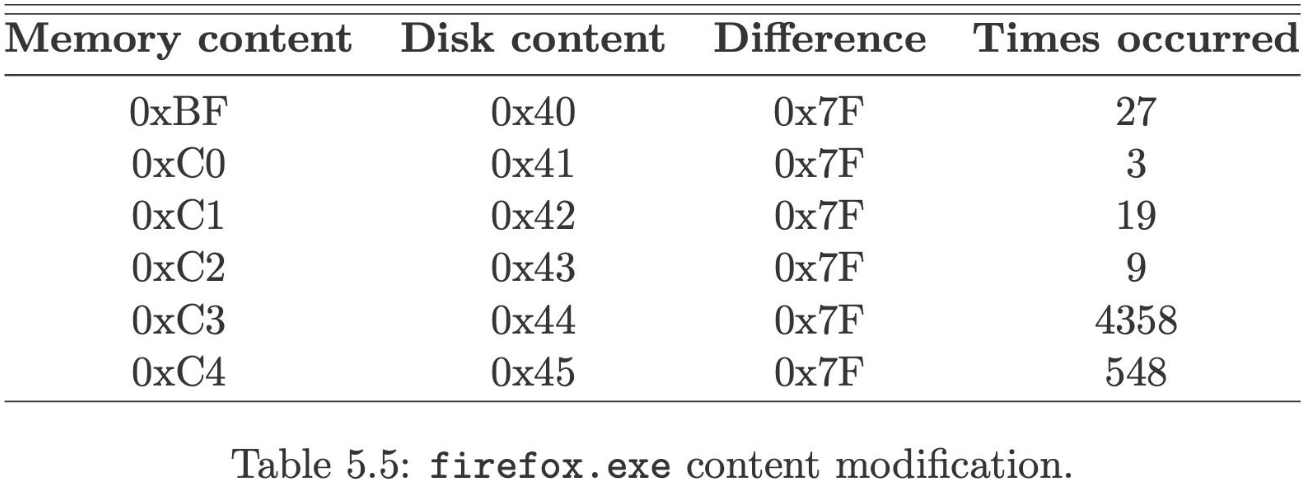


Figure 5.1: `firefox.exe` code segment bitmap.

False positives: Legitimate software



But they behave “well”:



Memory content	Disk content	Difference	Times occurred
0xBF	0x40	0x7F	27
0xC0	0x41	0x7F	3
0xC1	0x42	0x7F	19
0xC2	0x43	0x7F	9
0xC3	0x44	0x7F	4358
0xC4	0x45	0x7F	548

Table 5.5: `firefox.exe` content modification.

False positives:
Legitimate software

On *firefox.exe*:

- all the changes are one byte long.
- Mem: 89d1ebd4ff15b8e7c300cccccccccccccc
- Disk: 89d1ebd4ff15b8e74400cccccccccccccc
- The bytes ff15b8e7xx00 are repeated all over the place.

Other processes have similar behaviors.

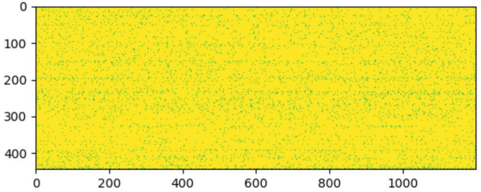
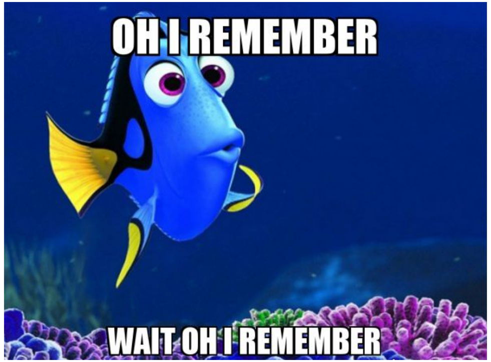


Figure 5.1: firefox.exe code segment bitmap.

Limitations

- Timing issues
  - To some extent, it’s a disadvantage of the live memory forensics: if the detective cannot arrive on time then it is likely that the evidence will not be there.
  - Accessing disk is slow.
- Size of malware families set.

In summary, what was the talk about?



- I present a technique for detecting fileless malware, especially process hollowing and process injection attacks.
- The results were promising, with a sensitivity rate of 92.11%.

And now, what do we do?

- Continue to investigate the false positive rate in order to reduce it.
- Mitigate the timing issues.
- Increase the number of malware families analyzed to generate more sound results.

Thanks!

Gracias!