Membrane: A Posteriori Detection of Malicious Code Loading by Memory Paging Analysis

Gábor Pék
Laboratory of Cryptography and System Security (CrySyS Lab)
Budapest University of Technology and Economics
www.crysys.hu

this is joint work with **Zsombor Lázár, Zoltán Várnagy, Márk Félegyházi , and Levente Buttyán**

Malicious code loading

It’s still a threat

↓

A ***wide range of techniques***

Existing detection approaches

Focus on the code loading technique

Evasion is easy, ***attacks persist*** for a massive period of time

The idea is en route

Code loading from the OS's perspective

Idea: focus to the anomaly

After loading a code the OS needs to
behave differently

On-demand paging in Windows

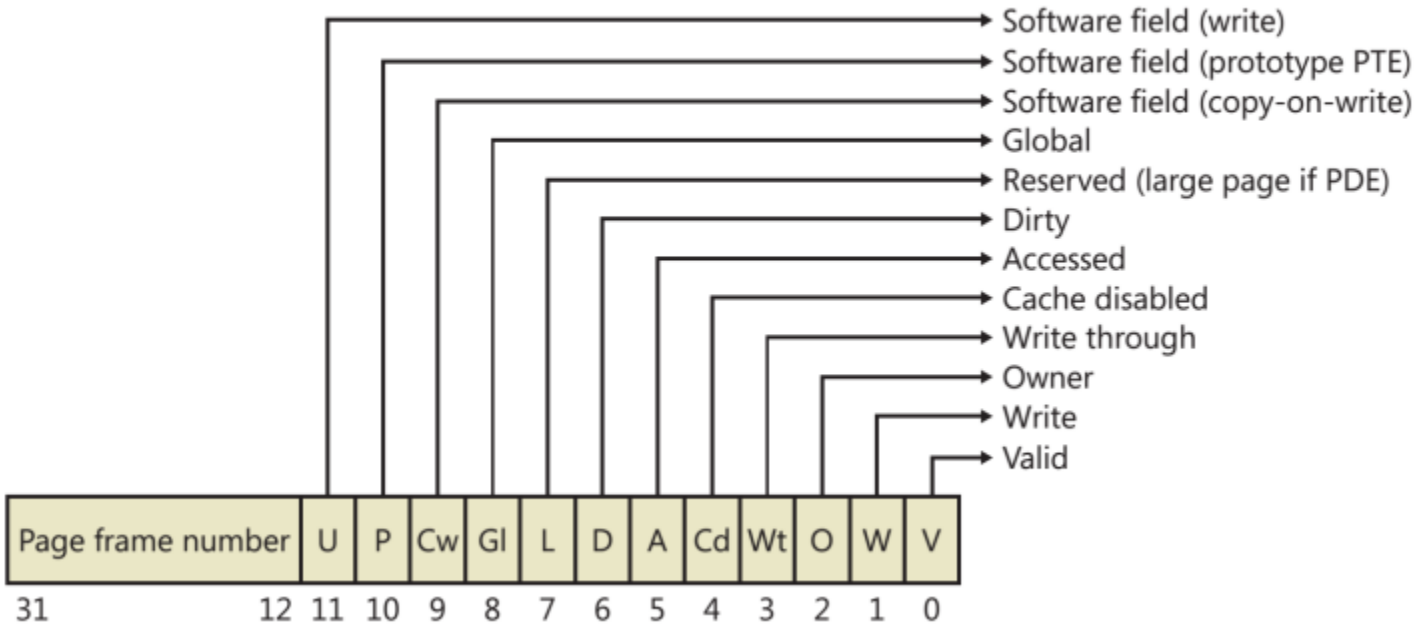
Lazy memory managment

Hardware page tables are built ***on-demand***

(stateful operation)

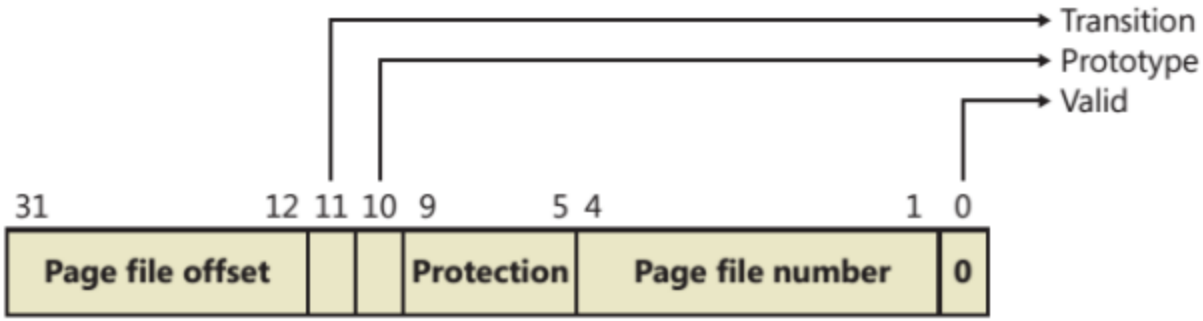
Hardware vs. Software MMU

Valid x86 hardware PTE



Mark E. Russinovich, David A. Solomon, Alex Ionescu: Windows Internals

x86 software PTE pointing to a page in a pagefile



Mark E. Russinovich, David A. Solomon, Alex Ionescu: Windows Internals

Example for a paging state

Demand Zero (Dz)

The MMU needs to provide a

zeroed out page.

Yes…!!! Another idea is coming

Calculate per process ***paging state cardinalities***

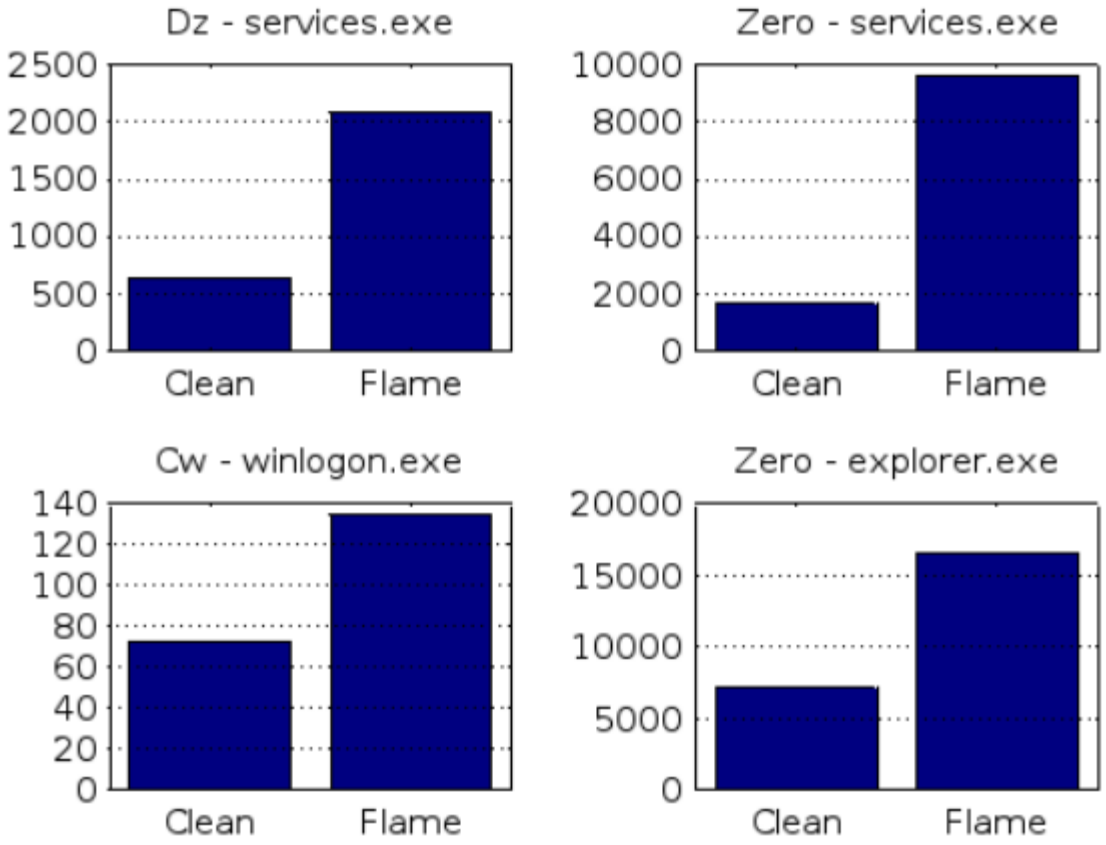
from

memory snapshots!

But how?

Let’s extend Volatility!

Cardinality of different paging states after Flame



Okay, but why?

No matter what technique is used

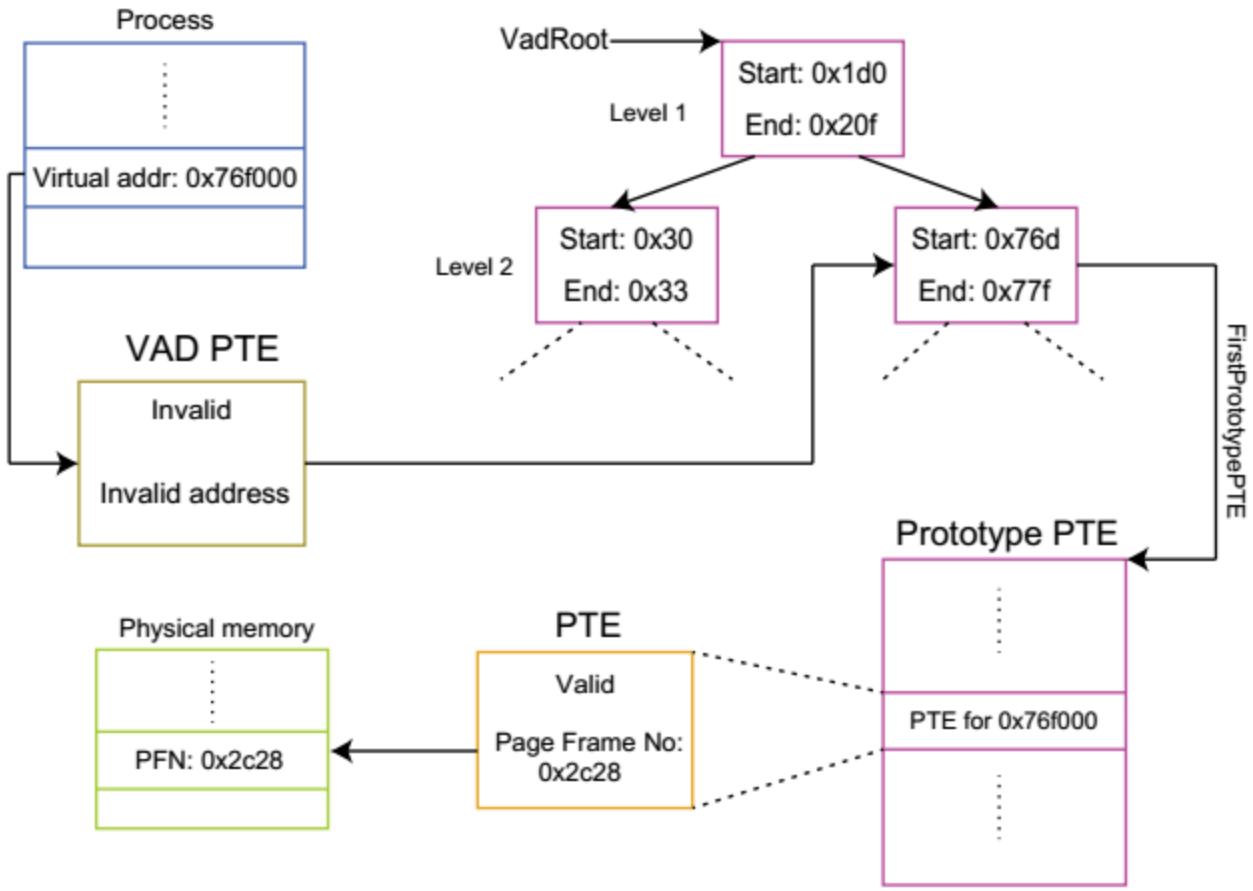
The loaded functionality ***distorts the paging state cardinalities*** of the original process

Collect paging states

We defined ***28 paging states***

(by exploring the realm of Windows)

Membrane as a Volatility extension

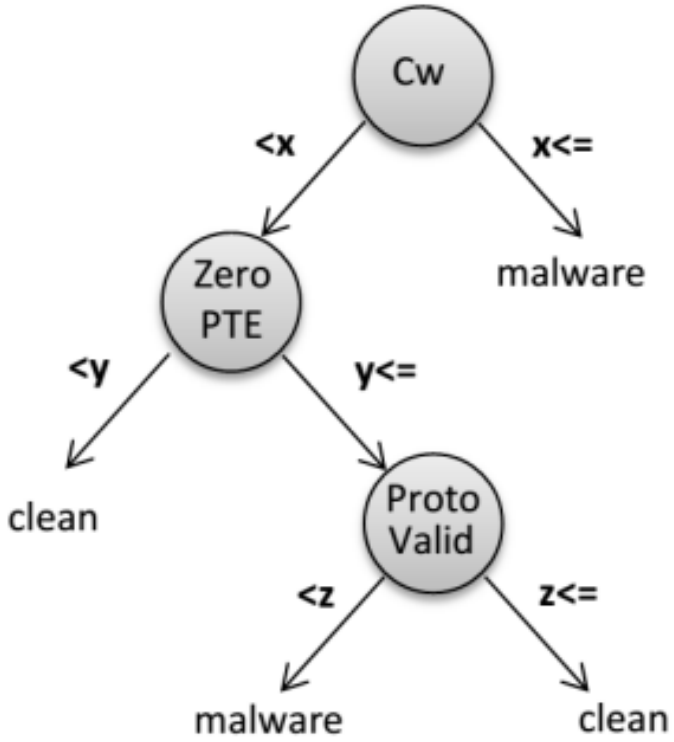


What if…machine learning is a fit now?

Accumulate ***per process paging state cardinalities*** into a ***feature set***

Use ***machine learning*** to select prominent features

Decision tree used by the random forest classifier

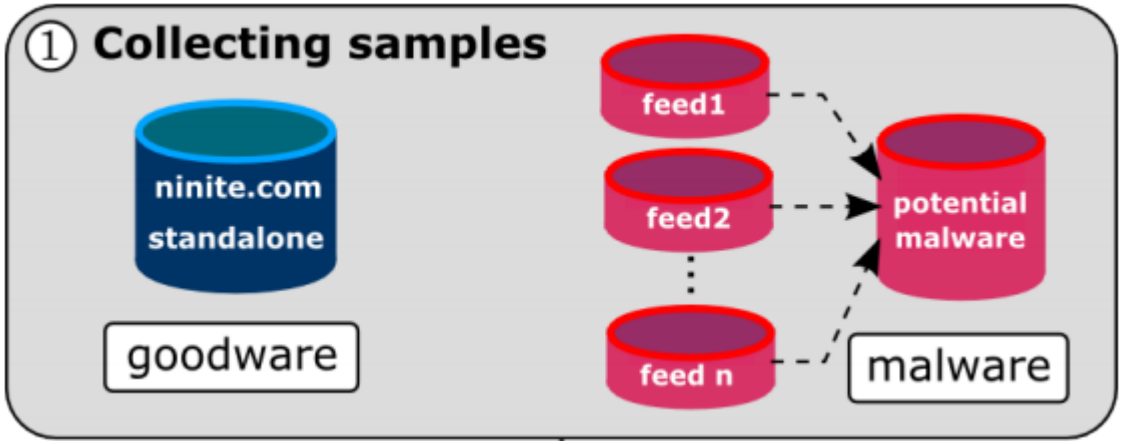


The idea seems to be complete…

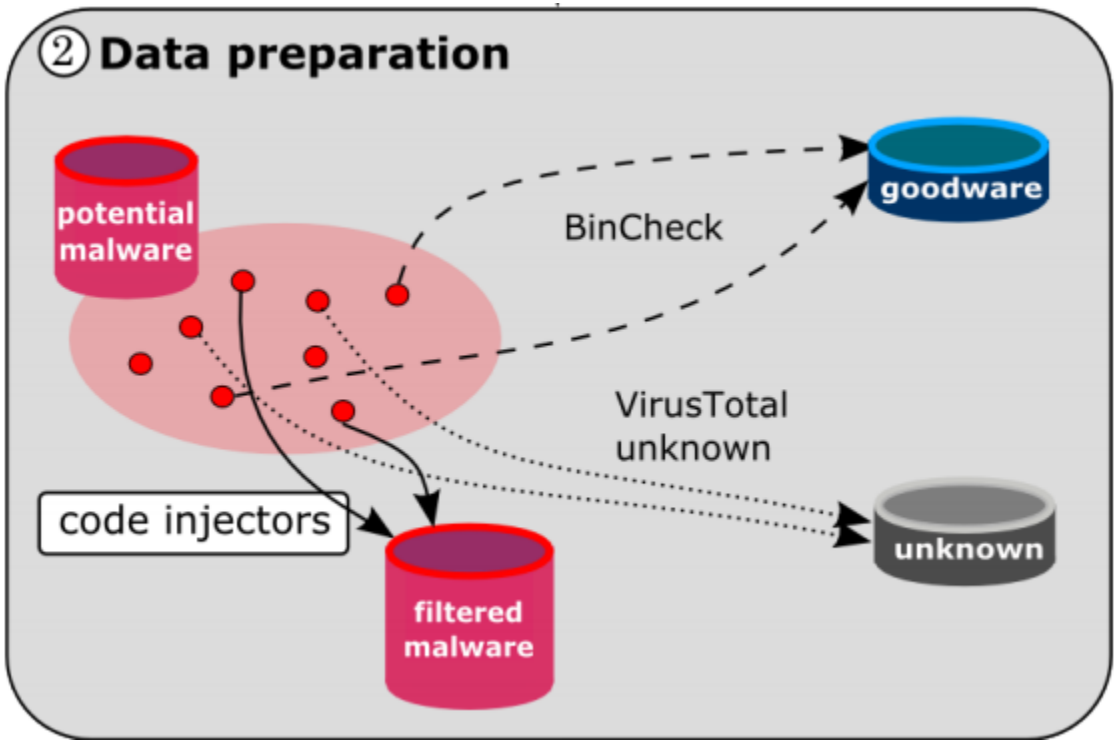
Oh my…

but we need a good corpus to do that

Methodology – #1 Collecting samples



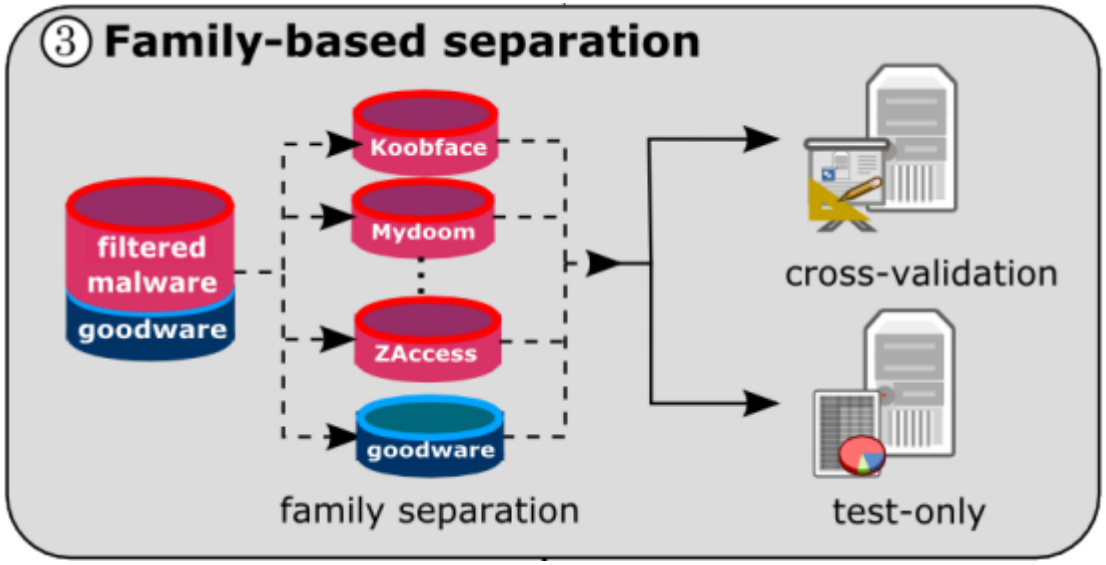
Methodology – #2 Data preparation



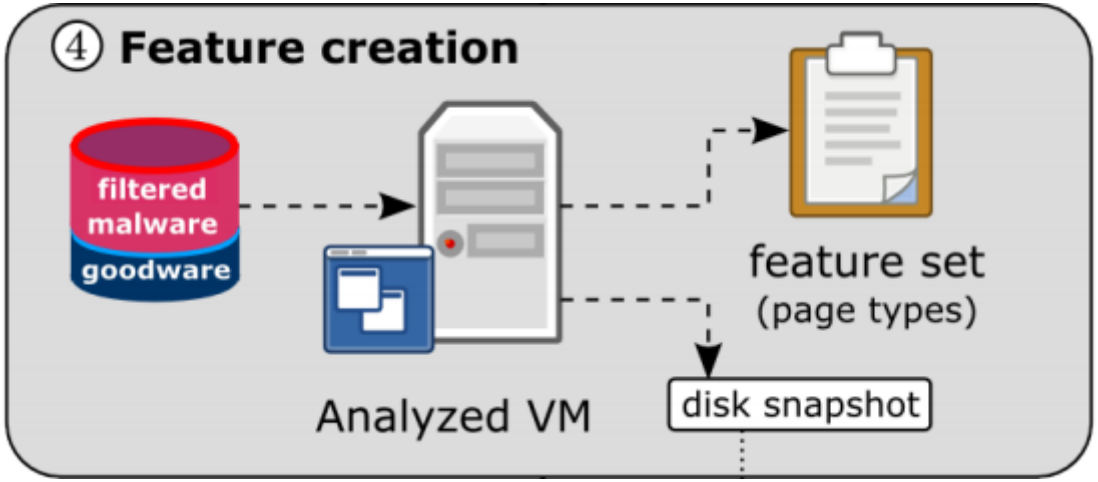
Preparing dataset

Stage	Count
Select malware samples	9,905
Remove goodware with BinCheck	9,903
Remove unknown with VirusTotal	9,121
Balance families	1,095
Select active + hide artifacts	658
Select code injecting malware	**194**

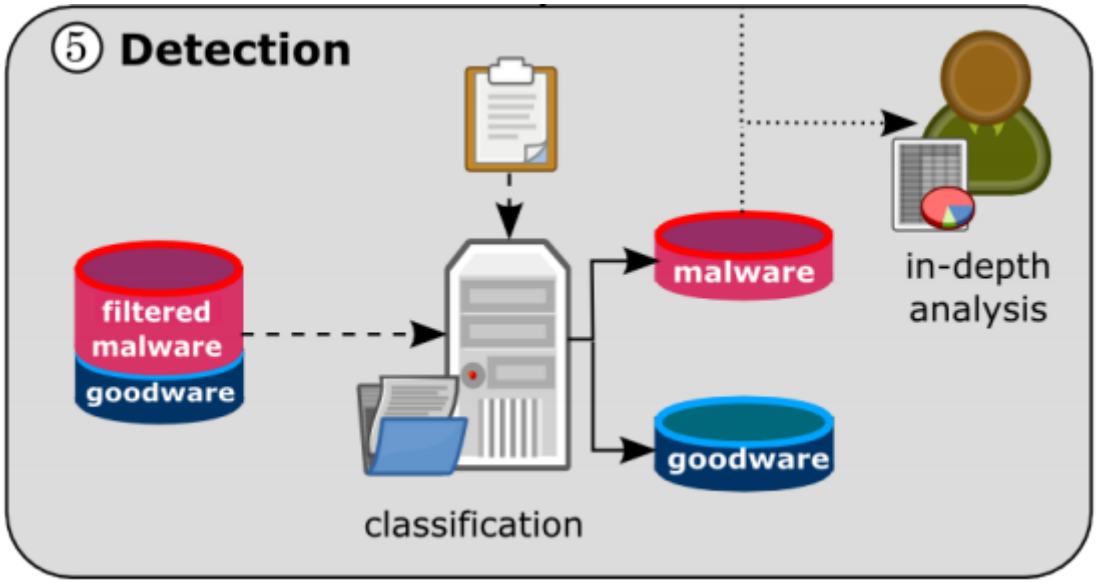
Methodology – #3 Family-based separation



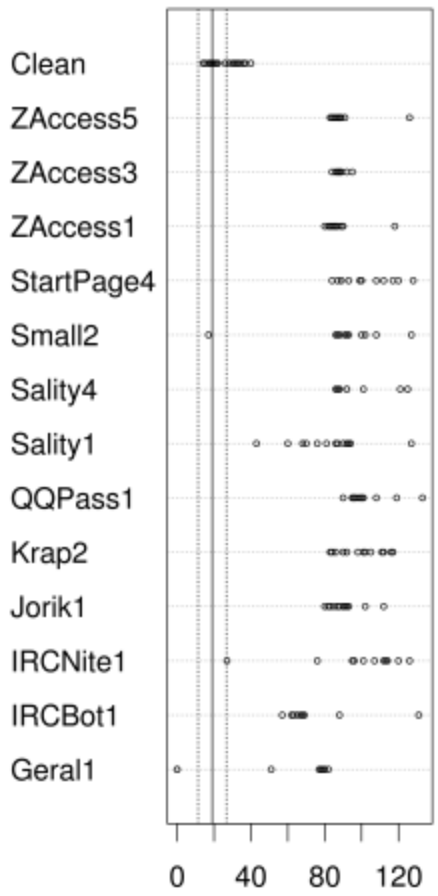
Methodology – #4 Feature creation



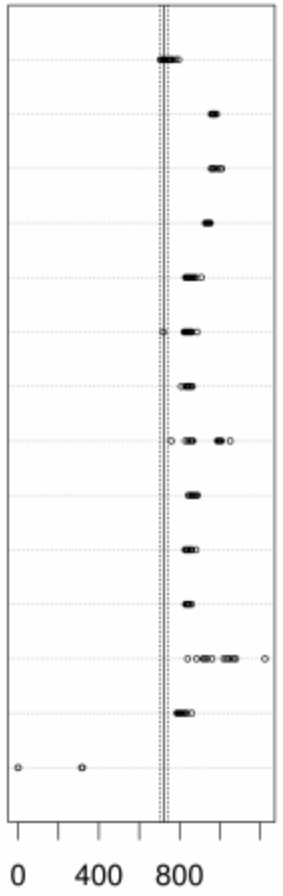
Methodology - Detection



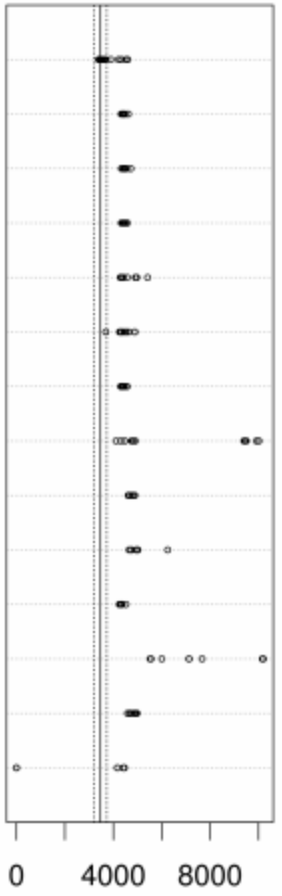
Birds-eye-view analysis of explorer.exe



(a) Copy-on-write



(b) DemandZero



(c) Zero PTE

Detecting generic and targeted malware in exp.exe

Internet	VM	ACC %	TPR %	FPR %
CROSS-VALIDATION DATASET				
no	WinXP	98.67	98.82	1.18
	Win7_1	73.59	75.92	28.46
yes	WinXP	92.81	90.88	5.45
	Win7_1	79.45	82.69	23.59
CROSS-VALIDATION DATASET WITH NO ADDITIONAL PROCESSES				
no	Win7_1	77.16	86.00	34.73
TEST-ONLY DATASET				
no	WinXP	100	100	-
	Win7_1	100	100	-

Membrane Live

Snapshot creation can be a ***heavy-duty operation***

We created a live monitoring version by extending a VMI framework

Questions?

```
git clone https://github.com/CrySyS/membrane.git
```