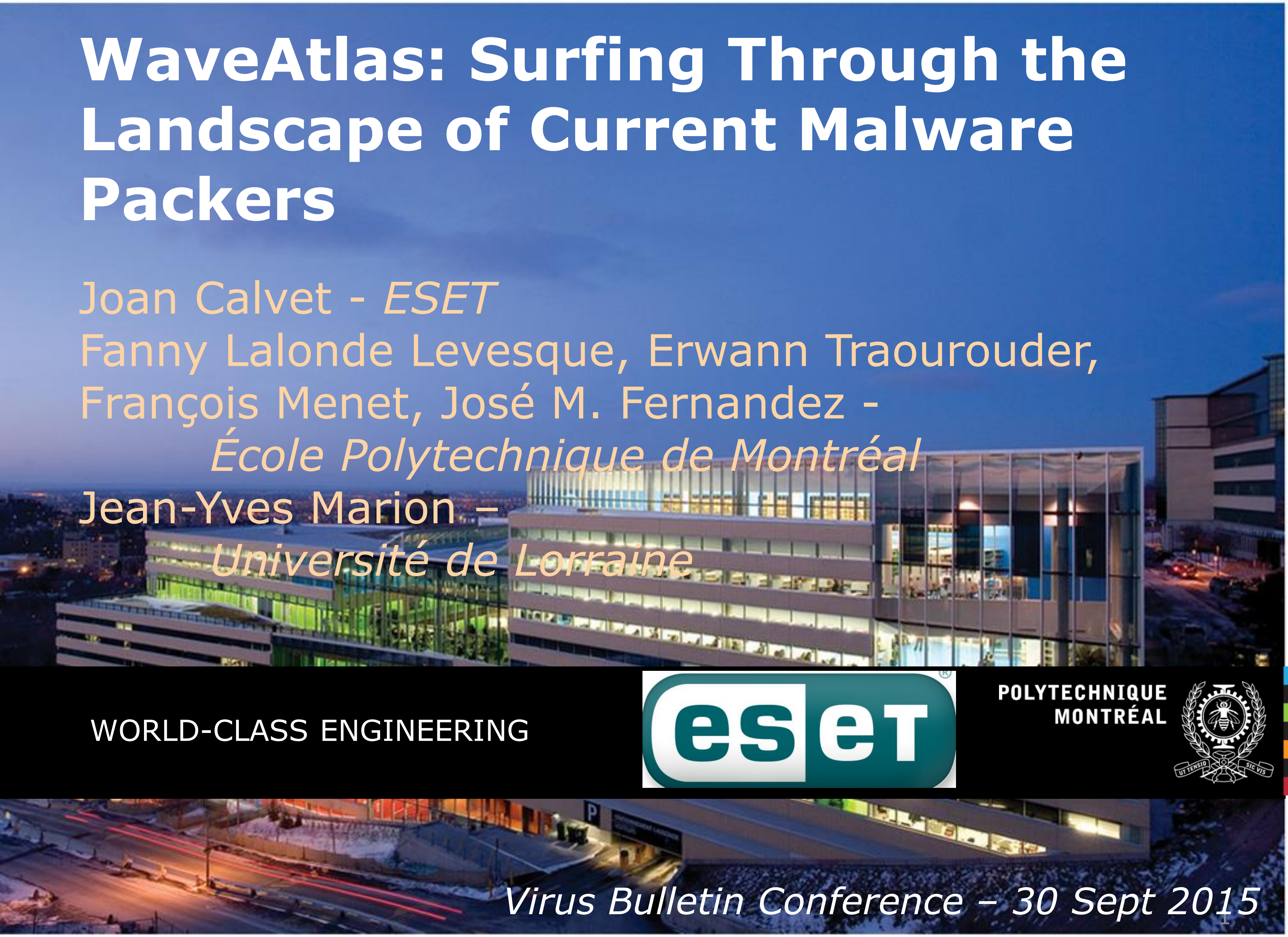


WaveAtlas: Surfing Through the Landscape of Current Malware Packers

Joan Calvet - *ESET*
Fanny Lalonde Levesque, Erwann Traourouder, François Menet, José M. Fernandez - *École Polytechnique de Montréal*
Jean-Yves Marion – *Université de Lorraine*

WORLD-CLASS ENGINEERING

POLYTECHNIQUE MONTRÉAL

Virus Bulletin Conference – 30 Sept 2015

OUTLINE

The question

The experiment

The results

Conclusions

RESEARCH QUESTION

Packer = *program transformation whose output is a program that will decrypt or decompress part of its code during its execution*

There is a wide range of techniques used by packers to protect their output program

Question: Is there a *packer model* ---as in, a set of features--- particularly prevalent among malware?

WHY DOES IT MATTER?

Exhibiting **common features** of malware packers would allow defenders to design suitable solutions

- Tailored program analysis for packers

Exhibiting **differences** in features of malware packers could enable classification and attribution

- Same packing tools will have same patterns
- Same groups will use same tools & techniques

RELATED WORK

Most of the existing work falls in two categories:

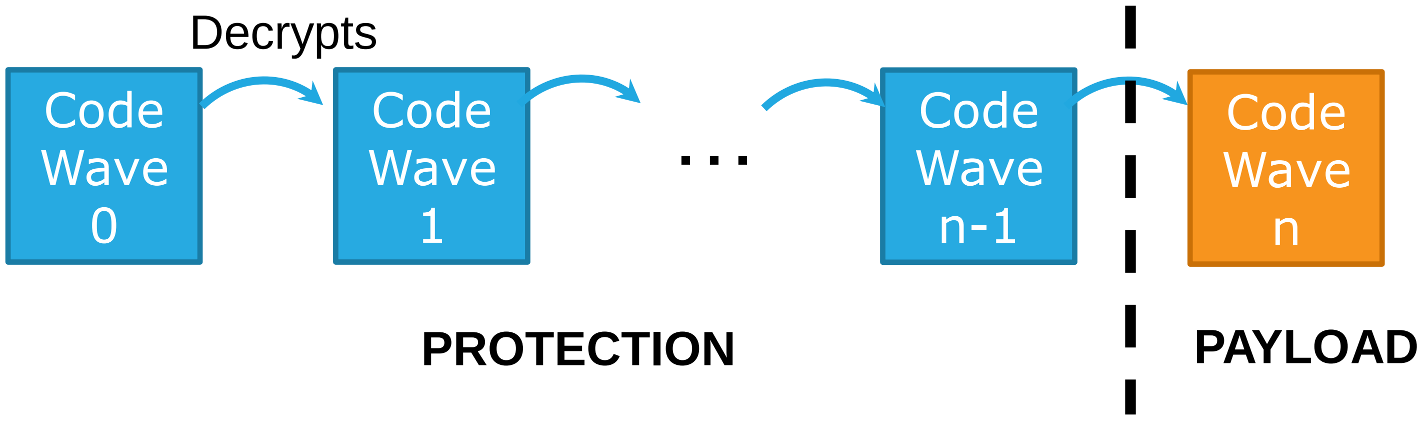
1. Statistics on packers built with static tools (PEiD), that provide names of known packers, but say nothing on their actual behaviour (or on unknown packers)

2. Technical descriptions of unusual packers, that do not help to draw a global picture

RESEARCH HYPOTHESIS

The following packer model *is* prevalent among malware:

- The execution passes through layers of self-modifying code (called *code waves*)
- The first code waves serve only to protect
- The last code wave contains the malware payload



So, how do we (dis)prove this hypothesis?

WAVE ATLAS: EXPERIMENTAL PROCEDURE AND SETUP

SAMPLE SELECTION (1)

We can not analyze all available malware samples

The significance of our experimental results will then depend on our choice of sample to study

To highlight global trends we focus on “successful” malware families (from their authors’ point-of-view…)

SAMPLE SELECTION (2)

Anti-viruses malware rankings are based on the number of samples, which artificially inflates the importance of file infectors

Instead, we measure malware success from the **response from the security community:**

- Idea: the more the community cares about a malware, the more impact the malware had on customers!
- Practical measurements: number of publications and take-down attempts

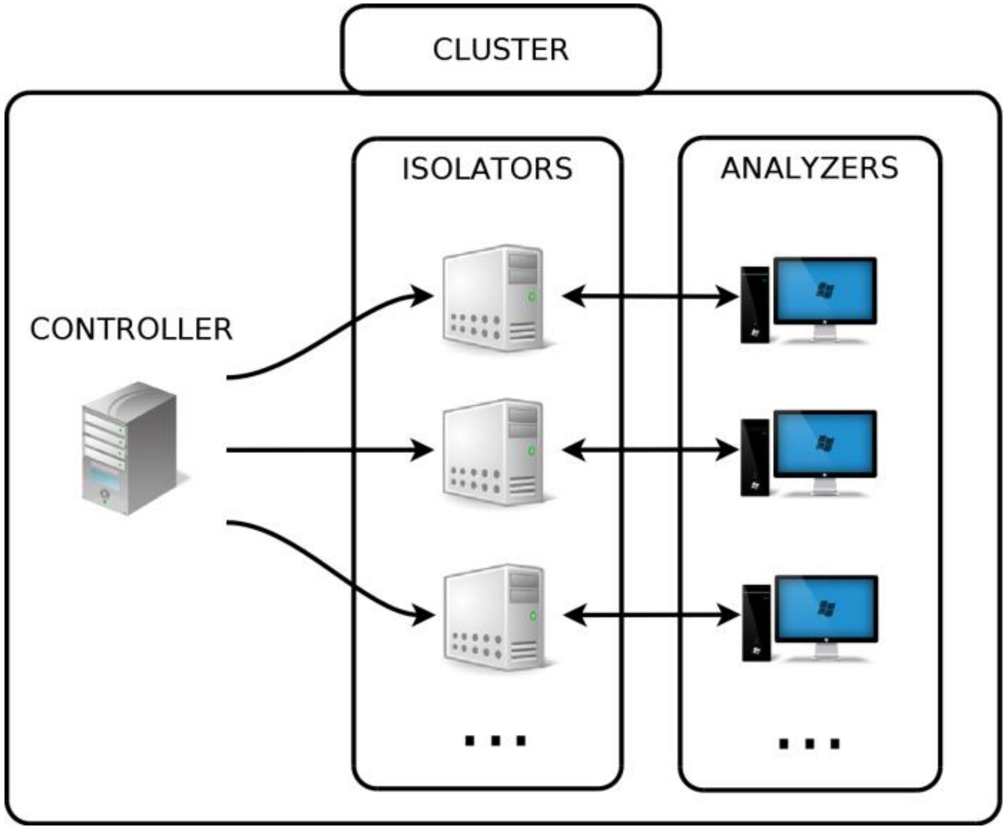
SAMPLES SELECTION (3)

Based on those criteria, we selected the following crimeware families:

Koobface, Conficker, Cutwail, Ramnit, Storm Worm, Waledac

For each of them, we analyzed **100 samples recognized as a member of the family by at least four major antivirus**

EXPERIMENTAL SETUP



Analyzers: execute malware sample for a maximum period of 20 minutes and collect a <u>detailed execution trace</u>

Isolators: answer network requests coming from associated analyzer with a network simulator

Controller: deploys samples and centralize results

EXPERIMENT VALIDATION

A malware sample does not necessarily yield valid experimental results, for example:

- It could be a corrupted file
- It could detect our environmental setup and refuse to execute normally

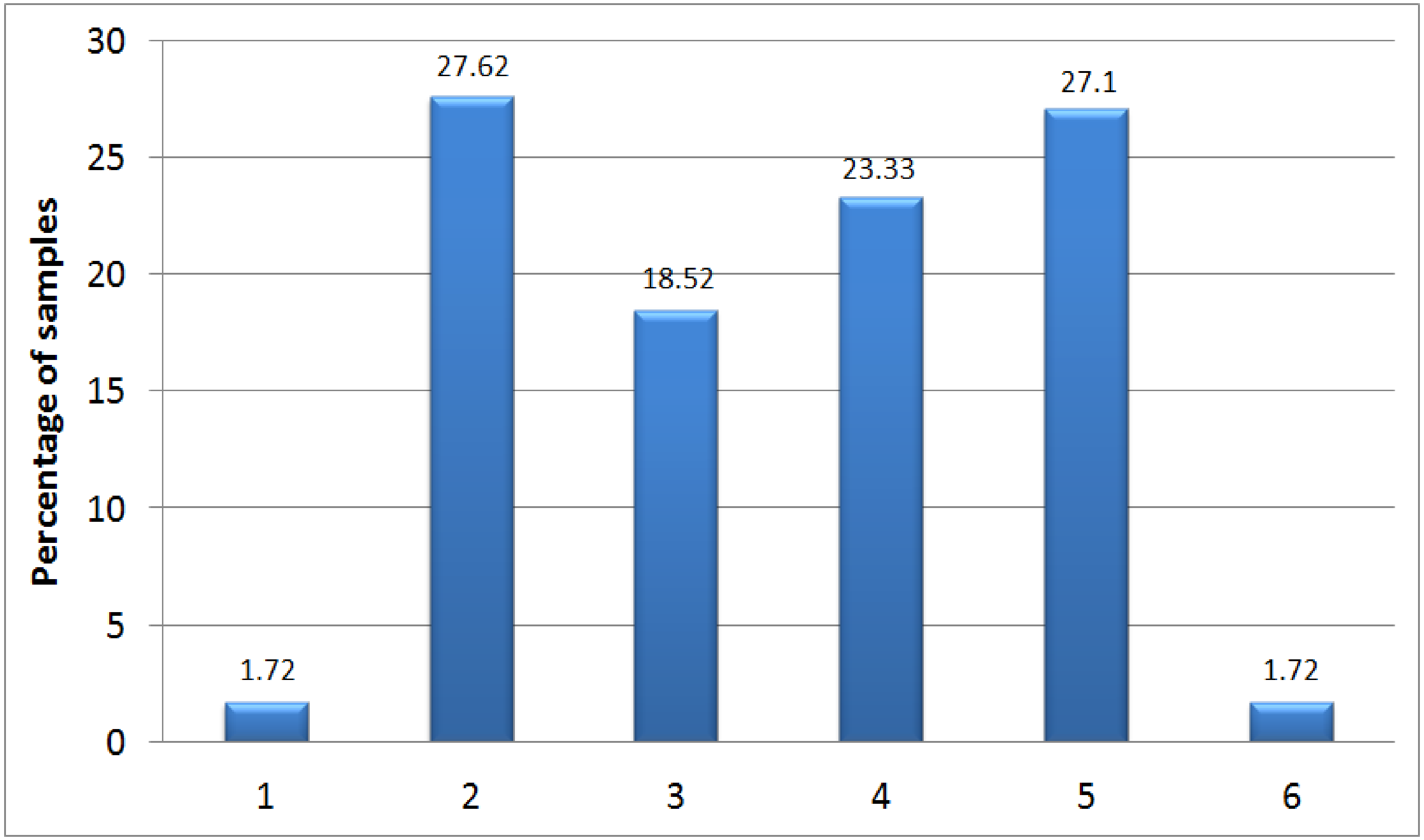
An experiment is said to be valid when:

- The execution trace contains more than 500 instructions, and
- The malware modified the system state during the execution (file creation, registry modification…)

We made sure to have 100 valid experiments for each selected family!

RESULTS

ARE OUR SAMPLES PACKED?



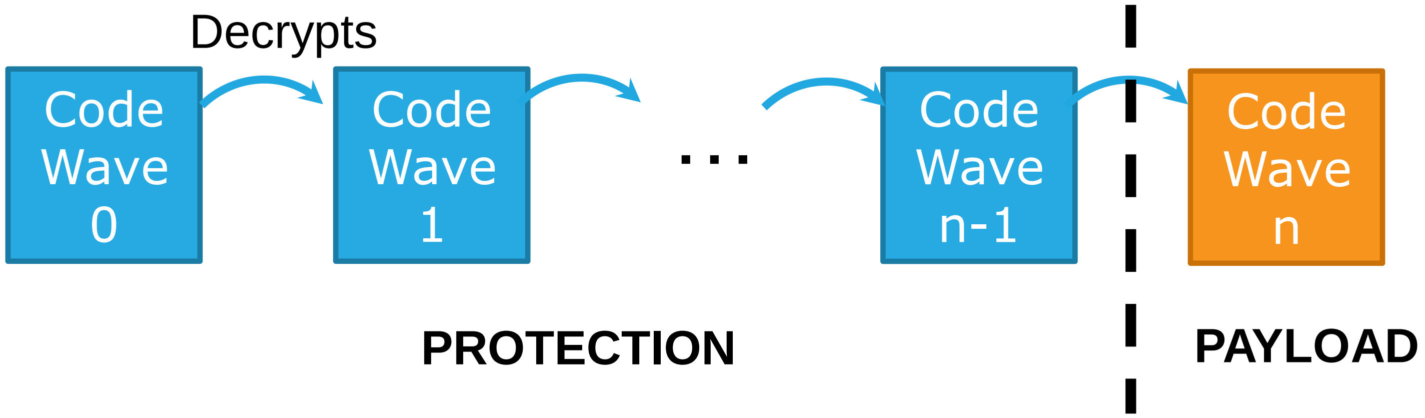
Number of code waves

REMEMBER OUR GOAL

We want to check if:

the first code waves (all except the last) serve to protect

the last code wave is the payload



MACHINE INSTRUCTION CHOICE

We established a list of classic assembly mnemonics from standard Windows programs

Those mnemonics are the ones chosen by compilers for performance reasons

We measured if a code wave executes more than 10 *non* classic mnemonics

Result: the first code wave is the only one to have exotic code in a significant number of samples

FUNCTION CALL CONVENTION

The CALL x86 instruction usually serves to implement function calls

It can also be used as a never returning branch instruction to make static analysis harder

We counted the number of “classic” uses of CALL and the number of never returning CALL

Results:

- **The first code waves have significantly less classic function calls than the last one**
- **The first code wave contains never returning CALL in a significant number of samples**

SYSTEM MODIFICATIONS

We measured in which code wave the malware interact with the system (file/registry modifications, network communications…)

Results:

- **In the majority of samples, the *last* code wave is the place where the malware modifies the system**

(More measurements & details in the paper)

HYPOTHESIS VALIDATION

- “*The first code waves serve only to protect*”
  - First code waves are particularly aggressive (exotic instructions, misused call conventions, exceptions thrown…)
  - First code waves do not interact with the system
- “*The last code wave contains the malware payload*”
  - Last code wave is standard code
  - Last code wave interacts with the system

SPECIAL CASES

In some samples, our hypothesis was actually only partially validated:

- The penultimate code wave is the malware payload
- The last code wave contains some hooks put in place by the payload

In those cases the limit between the protection and the payload is therefore the penultimate code wave

CONCLUSION

- Successful crimeware employs a simple packer model, where code layers are added around the payload code for protection

- Why after all these years do malware authors stick to this model?

- Can we do better as defenders?