# Challenges and results in automatic malware analysis and classification

Stefano Zanero, PhD

Assistant Professor, Politecnico di Milano

## Knowing the enemy = key for success

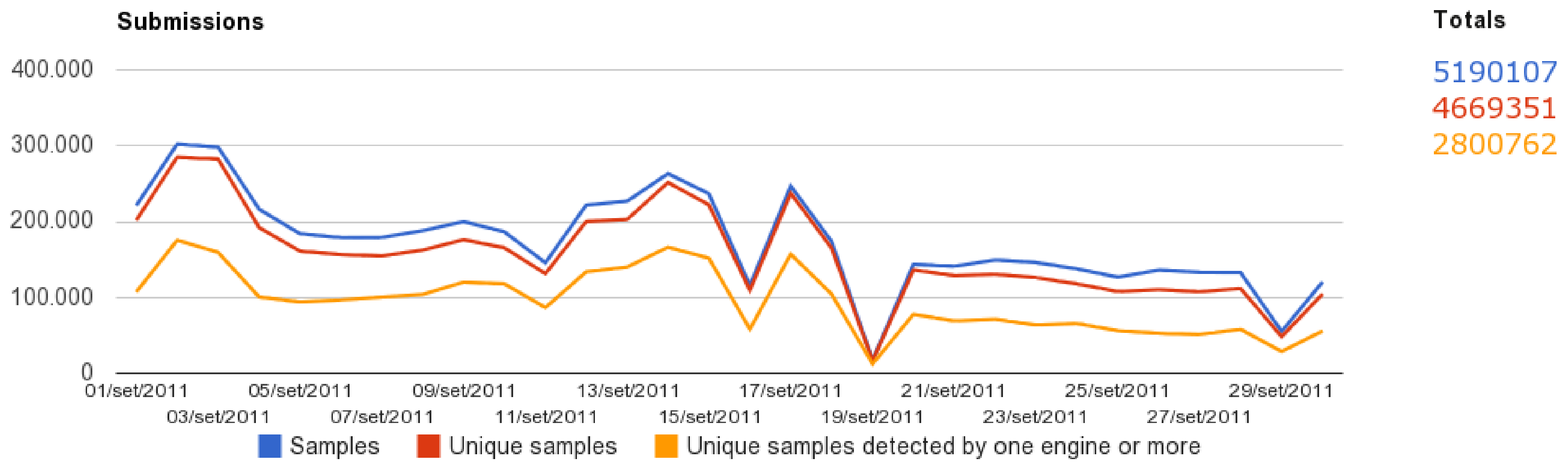
*“He will win who knows when to fight and when not to fight... He will win who, prepared himself, waits to take the enemy unprepared. Hence the saying: If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle.”* [Sun-Tsu]

## The malware problem

**Malware** at the root of many internet security problems

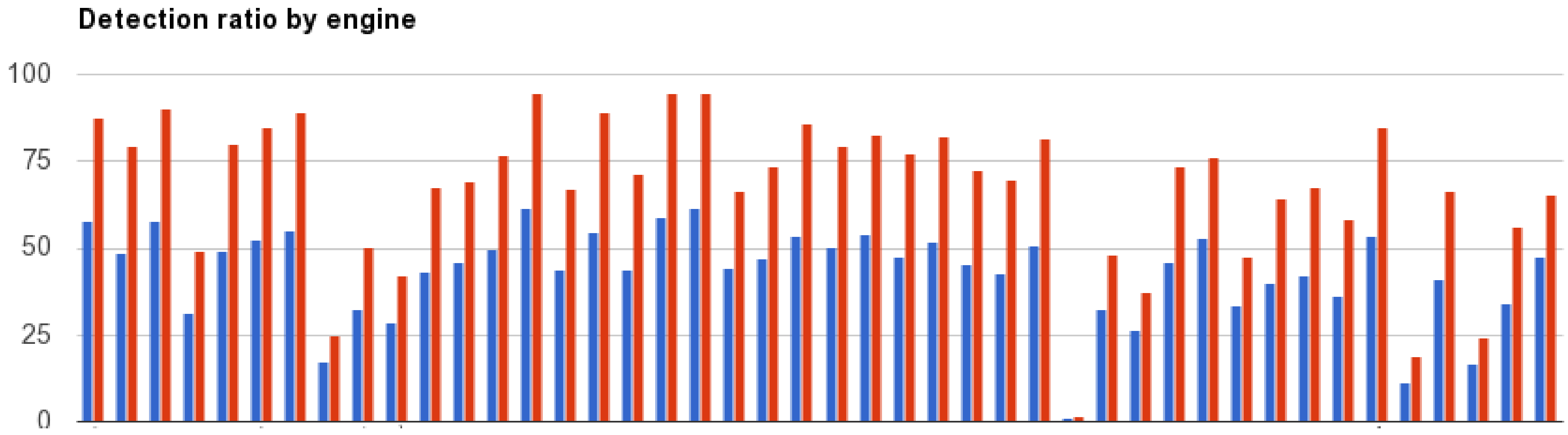
- Tens of thousands of new samples each day!
- Viruses developed with creation kits
- Underground economy fuelling malware creation
- 1990s: explosive diffusion of identical malware
- 2010s: stealthy diffusion of variants of malware designed to be difficult to identify, trace and

## Number of new, unique samples per day



Data related to september 2011
Thanks to VirusTotal (www.virustotal.com)

# Antivirus detection ratio...



## AV industry in 1998

## AV industry in 2008

## The analysis issue

Analysts are way too few, code is way too much

Need better ways to

- Automatically analyze/reverse engineer malware
- Automatically classify/cluster malware, e.g. in families

For both, we have two approaches with symmetric issues

## Static vs. dynamic approaches

**Static approaches**

+ Complete analysis

- Difficult to extract semantics

- Obfuscation / packing

**Dynamic approaches**

+ Easy to see “behaviors”

+ Malware unpacks itself

- “Dormant” code

## Our Approach

“Turn weakness into strength” (Sun-Tzu): leverage code reuse between malware samples to our advantage

- Automatically generate semantic-aware models of code implementing a given malicious behavior
- Use these models to statically detect the malicious functionality in samples that do not perform that behavior during dynamic analysis

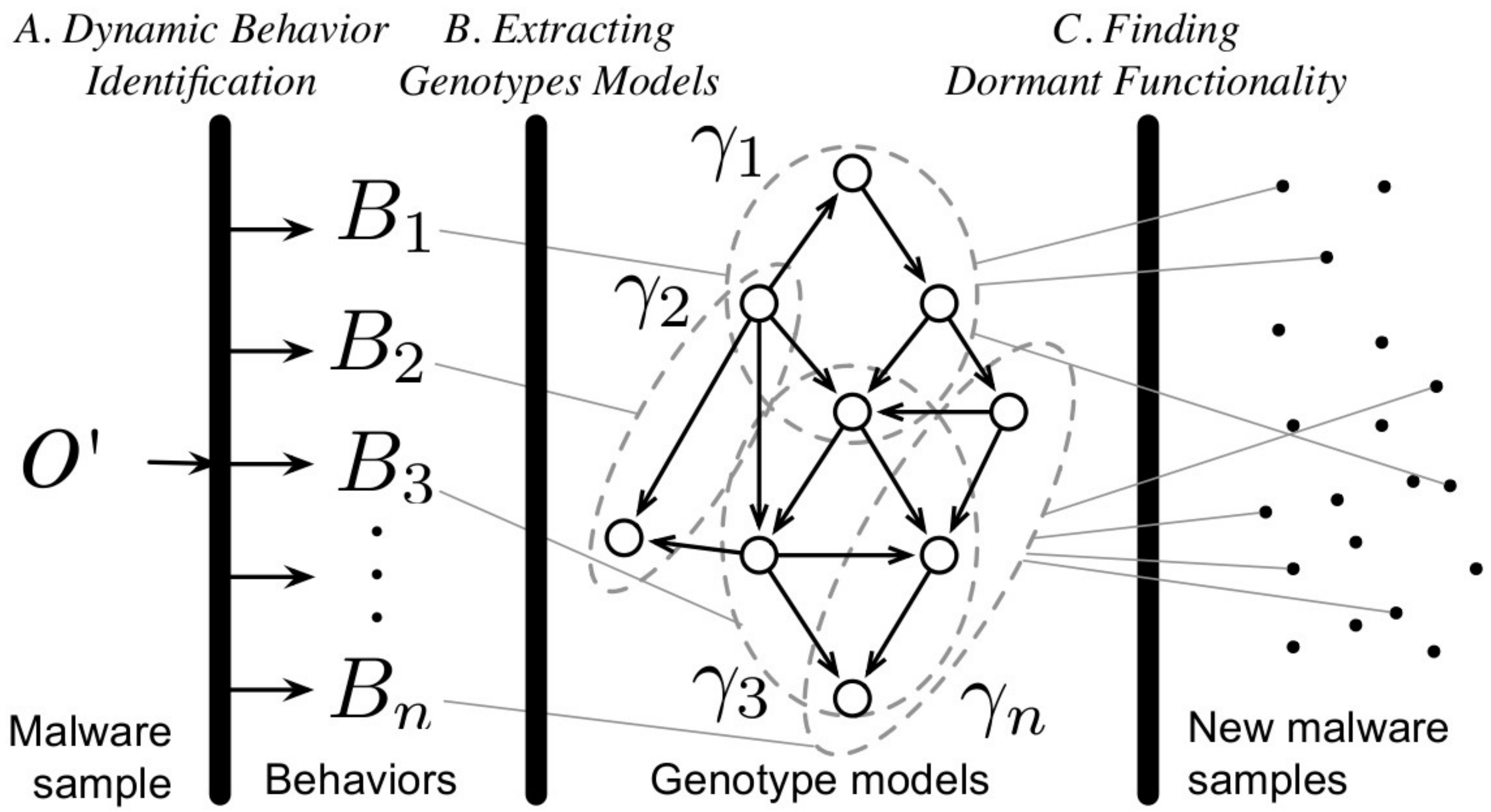
## REANIMATOR

Run malware in monitored environment and detect a malicious behavior *(phenotype)*

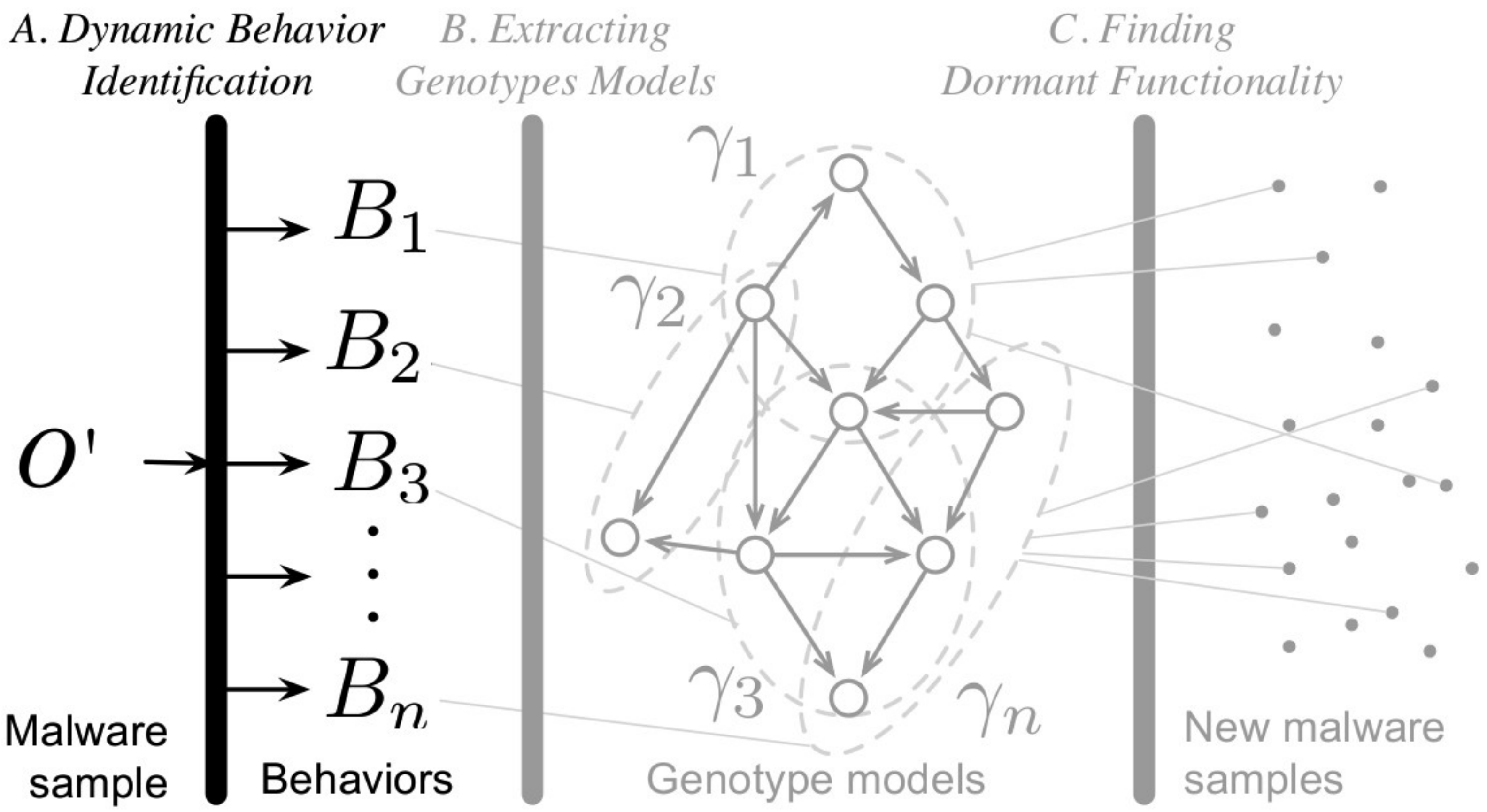
Identify and model the code responsible for the malicious behavior *(genotype model)*

Match genotype model against other unpacked binaries

# REANIMATOR


A. Dynamic Behavior Identification
B. Extracting Genotypes Models
C. Finding Dormant Functionality
$O'$
$B_1$
$B_2$
$B_3$
$B_n$
$\gamma_1$
$\gamma_2$
$\gamma_3$
$\gamma_n$
Malware sample
Behaviors
Genotype models
New malware samples

# Dynamic Behavior Identification



## Dynamic Behavior Identification

Run malware in instrumented sandbox

- Anubis (anubis.iseclab.org)

Dynamically detect a behavior B *(phenotype)*

Map B to the set $R_B$ of system/API call instances responsible for it

$R_B$ is the output of the behavior identification phase

## Behavior Detection Examples

**spam**: send SMTP traffic on port 25

- network level detection

**sniff**: open promiscuous mode socket

- system call level detection

**rpcbind**: attempt remote exploit against a specific vulnerability

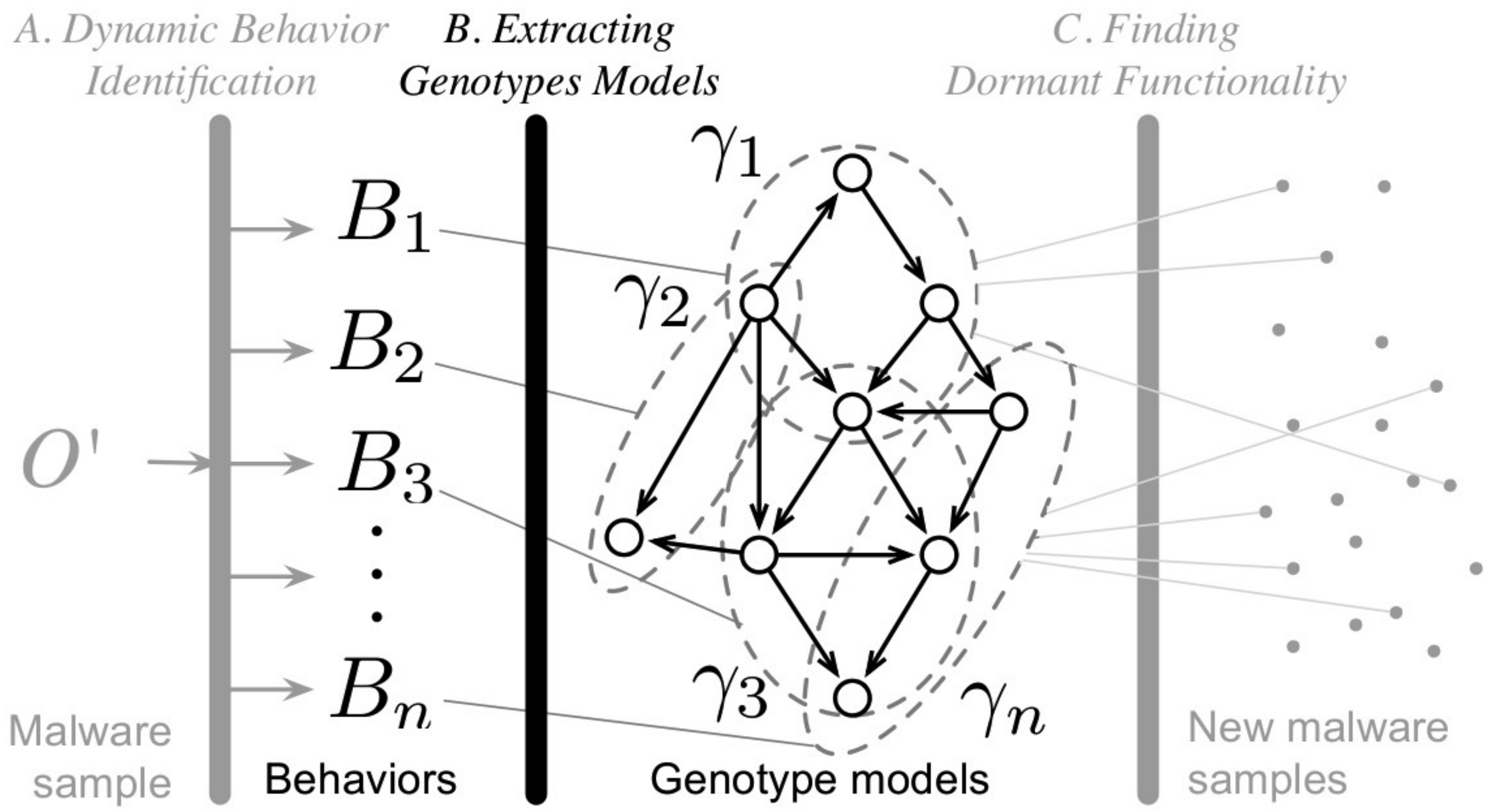
- network level detection, with snort signature

**drop**: drop and execute a binary

- system call level detection, using data flow information

**...**

## Extracting Genotype Models



## Extracting Genotype Models: Goals

Identified genotype should be precise and complete

- Complete: include all of the code implementing B
- Precise: do not include code that is not specific to B (utility functions,..)

We proceed by *slicing* the code, then *filtering* it to remove support code, and *germinating* to complete it

## Slicing

Start from relevant calls $R_B$

Include into slice $\phi$ instructions involved in:

- preparing input for calls in $R_B$
  - follow data flow dependencies backwards from call inputs
- processing the outputs of calls in $R_B$
  - follow data flow forward from call outputs

We do not consider control-flow dependencies

- would lead to including too much code (taint explosion problem)

## Filtering

The slice ϕ is not precise

General purpose utility functions are frequently included (i.e: string processing)

- may be from statically linked libraries (i.e: libc)
- genotype model would match against any binary that links to the same library

Backwards slicing goes too far back: initialization and even unpacking routines are often included

- genotype model would match against any malware packed with the same packer

## Filtering Techniques

Exclusive instructions:

- set of instructions that manipulate tainted data **every time** they are executed
- utility functions are likely to be also invoked on untainted data

Discard whitelisted code:

- whitelist obtained from other tasks or execution of **the same sample**, that do not perform B
- could also use foreign whitelist
  - i.e: including common libraries and unpacking routines

## Germination

The slice ϕ is not complete

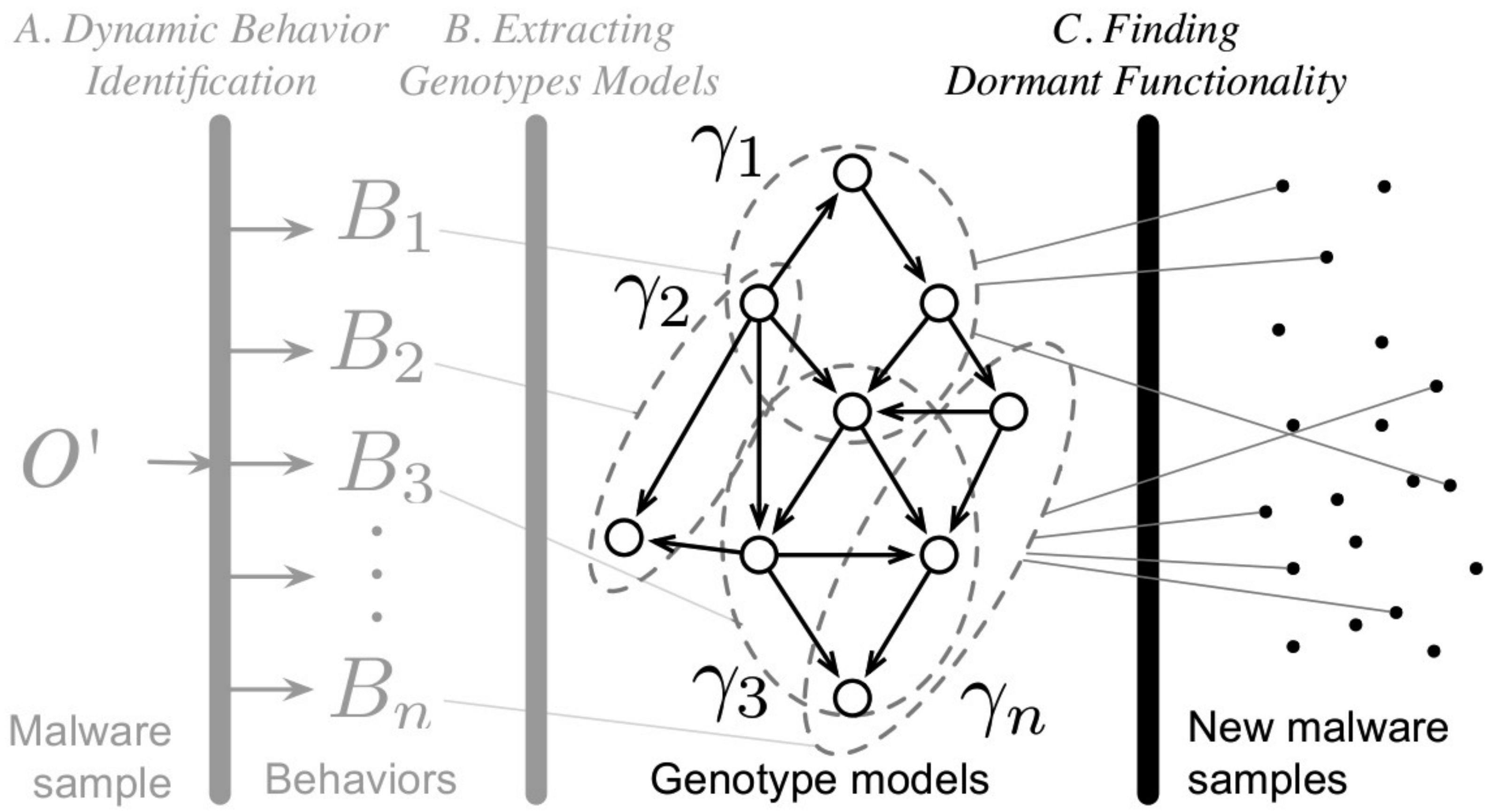
Auxiliary instructions are not included

- loop and stack operations, pointer arithmetic, etc

Add instructions that cannot be executed without executing at least one instruction in ϕ

Based on graph reachability analysis on the intra-procedural Control Flow Graph (CFG)

# Finding Dormant Functionality



## Finding Dormant Functionality

Genotype is a set of instructions

Genotype model is its colored control flow graph (CFG)

- nodes colored based on instruction classes

2 models match if they share at least one K-Node subgraph (K=10)

Use techniques by Kruegel et al. to efficiently match a binary against a set of genotype models

We use Anubis as a generic unpacker

## Evaluation

Are the results accurate?

- when REANIMATOR detects a match, is there really the dormant behavior?
- how reliably does REANIMATOR detect dormant behavior in the face of recompilation or modification of the source code?

Are the results insightful?

- does REANIMATOR reveal behavior we would not see in dynamic analysis?

## Accuracy

To test accuracy and robustness of our system we need a ground truth

Dataset of 208 bots with source code

- thanks to Jon Oberheide and Michael Bailey from University of Michigan

Extract 6 genotype models from 1 bot

Match against remaining 207 bot binaries

## Accuracy

Even with source, manually verifying code similarity is time-consuming

Use a source code plagiarism detection tool

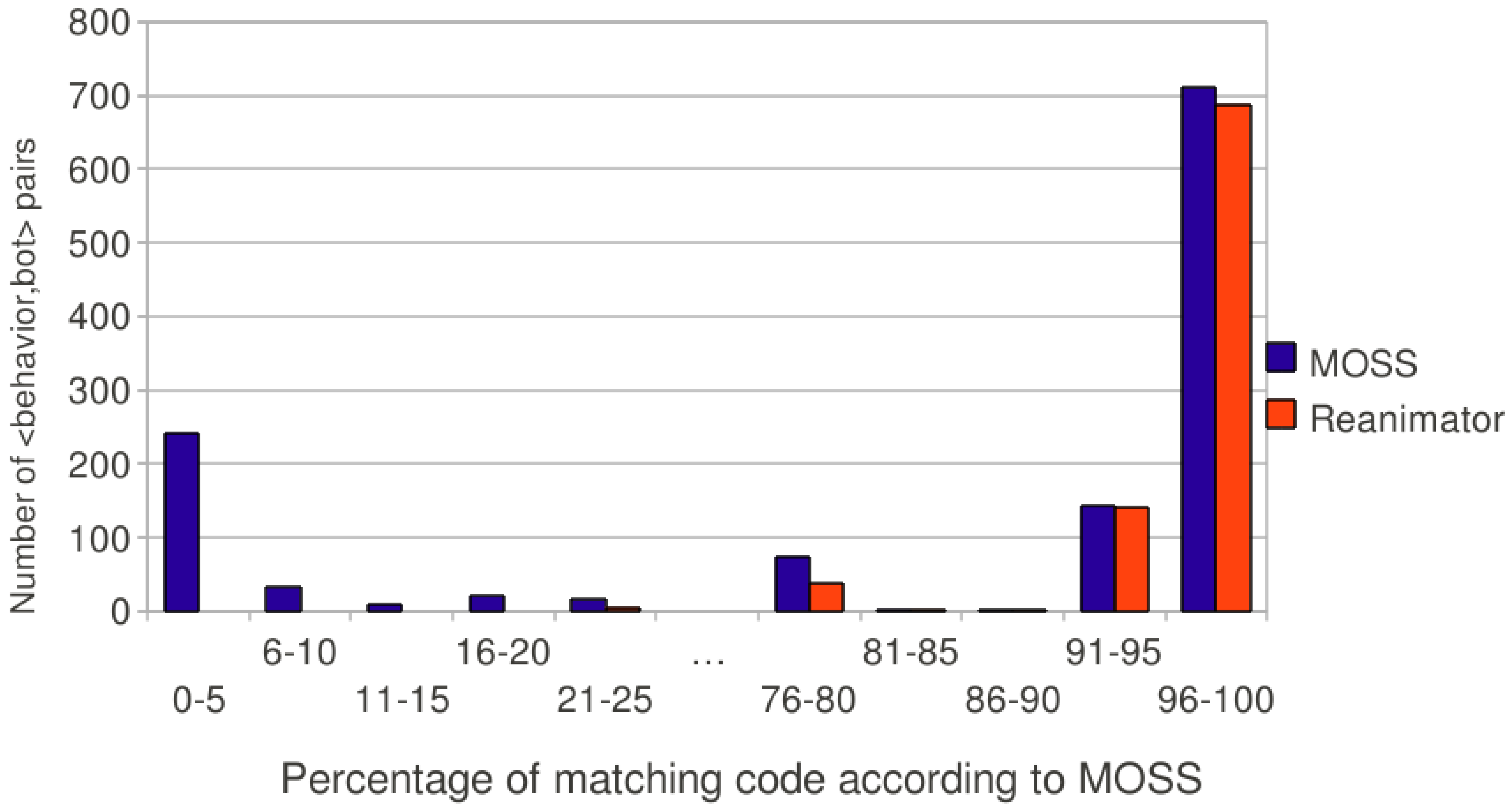
- MOSS

We feed MOSS the source code corresponding to each of the 6 behaviors

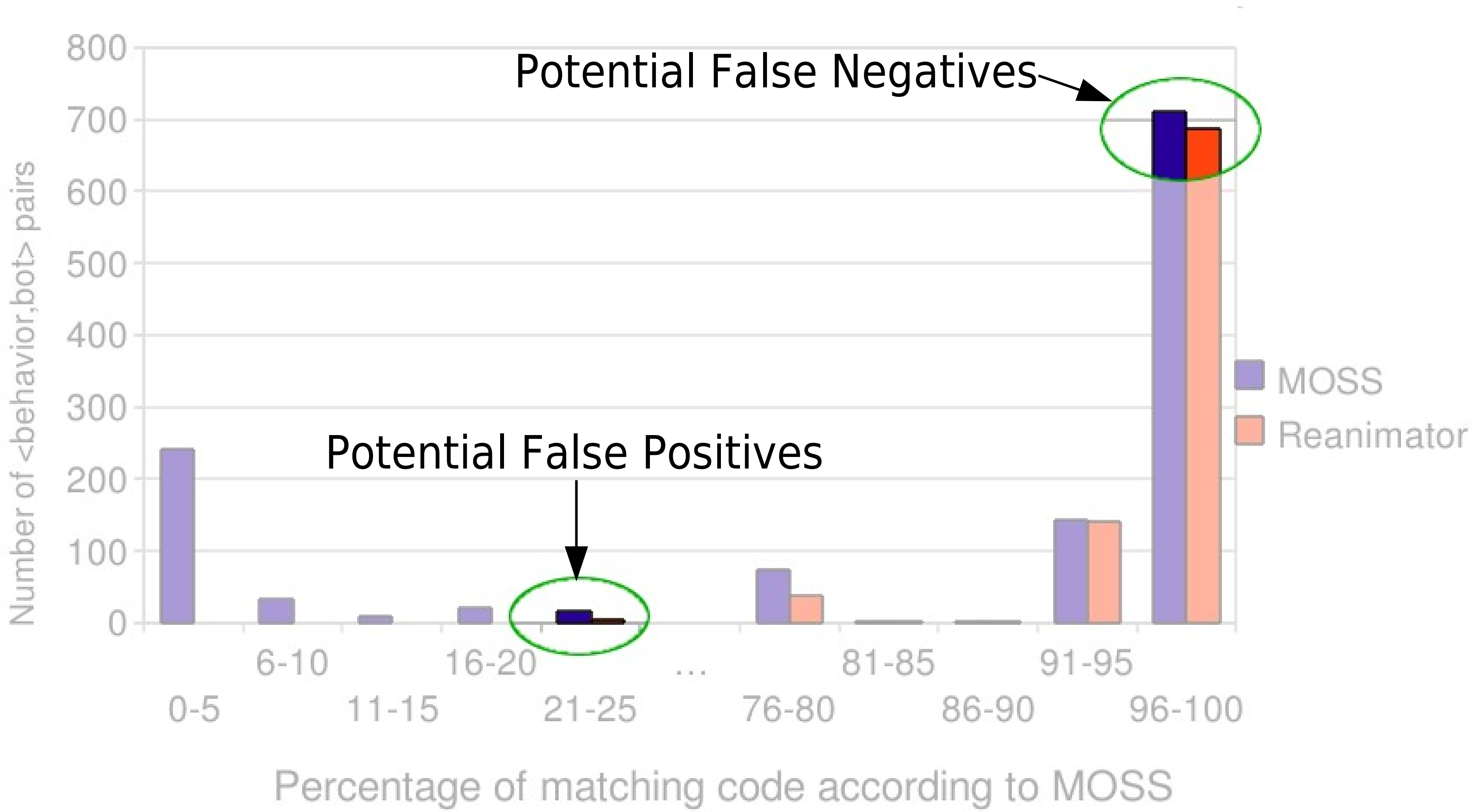
- match it against the other 207 bot sources
- MOSS returns a similarity score in percentage

We expect REANIMATOR to match in cases where MOSS returns high similarity scores

## MOSS Comparison



## MOSS Comparison



## Accuracy Results

We manually investigated the potential false positives and false negatives

Low false negative rate (~1.5%)

- mostly small genotypes

No false positives

- genotype model match always corresponds to presence of code implementing the behavior

Also no false positives against dataset of ~2000 benign binaries

- binaries in system32 on a windows install

## Robustness

Robustness results when re-compiling same source

- Robust against different compilation options (<7% false negatives)
- Robust against different compiler versions
- Not robust against completely different compiler (>80% false negatives)
- Some robustness to malware metamorphism was demonstrated by Kruegel in a previous work

## In-the-Wild Detection

10 genotype models extracted from 4 binaries

4 datasets

- irc_bots: 10238 IRC bots
- packed_bots: 4523 packed IRC bots
- pushdo: 77 pushdo binaries (dropper, typically drops spam engine cutwail)
- allaple: 64 allaple binaries (network worm)

Reanimator reveals a lot of functionality not observed during dynamic analysis

# In-the-Wild Detection

| Genotype | Phenotype | irc_bots | | | | packed_bots | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | B | S | D | B ∩ S | B | S | D | B ∩ S |
| httpd | backdoor | 2014 | 636 | 635 | 279 | 840 | 425 | 425 | 264 |
| keylog | keylog | 0 | 293 | 254 | 0 | 0 | 120 | 111 | 0 |
| killproc | killproc | 0 | 400 | 400 | 0 | 4 | 62 | 62 | 0 |
| simplespam | spam | 154 | 409 | 409 | 0 | 53 | 204 | 204 | 0 |
| udpflood | packetflood | 0 | 374 | 342 | 0 | 0 | 139 | 122 | 0 |
| sniff | sniff | 43 | 270 | 72 | 0 | 120 | 204 | 45 | 0 |

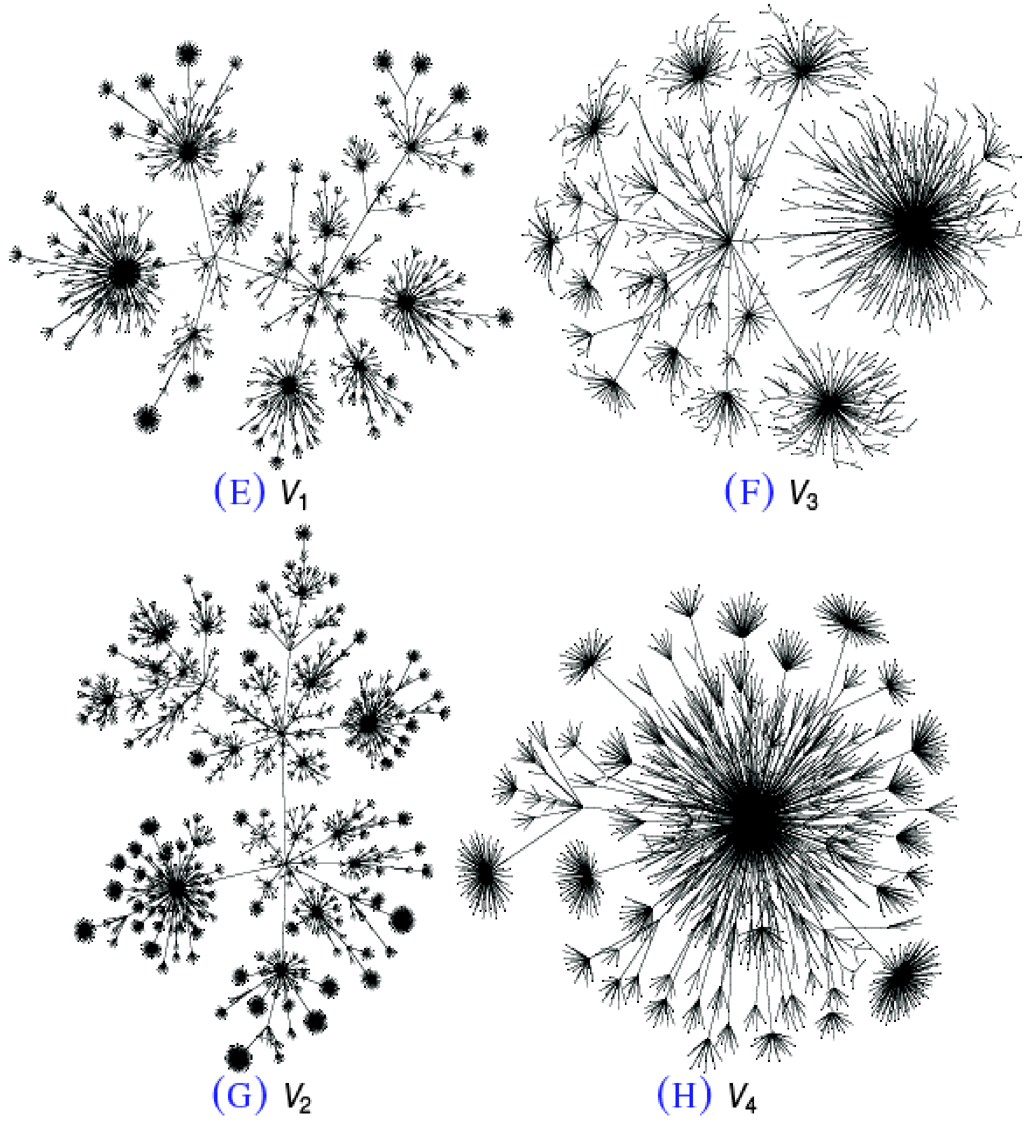
| Genotype | pushdo | | | | allaple | | | |
|---|---|---|---|---|---|---|---|---|
| | B | S | D | B ∩ S | B | S | D | B ∩ S |
| drop | 50 | 54 | 54 | 46 | 0 | 0 | 0 | 0 |
| spam | 1 | 43 | 42 | 1 | 0 | 0 | 0 | 0 |
| scan | 23 | 0 | 0 | 0 | 58 | 61 | 61 | 58 |
| rpcbind | 5 | 9 | 0 | 1 | 62 | 61 | 61 | 58 |

B: Behavior observed in dynamic analysis.

S,D: Functionality detected by Reanimator

## Malware classification

- An open problem with much confusion
- Classification by antivirus vendors completely unreliable
- We demonstrated this by analyzing naming inconsistencies among them
- Many **strong** inconsistencies which cannot be solved by simply remapping names



# Classifying malware by structural and behavioral features

Several works perform either:

- Structural clustering based on code features (e.g. works by H. Flake, Ero Carrera, and others)
- Behavioral clustering based on program execution traces (e.g. works by P. M. Comparetti, C. Kruegel, and others)

Our current research: using the same backward-forward techniques we used in Reanimator to map these two clustering approaches to each other. This will improve the quality of the families, help cluster correctly malware which is obfuscated or which has dormant behaviors

## Thanks for your attention!

Any question unanswered during Q&A or any follow up:

stefano.zanero@polimi.it

Most of the work presented was/is joint work with:

UCSB – Christopher Kruegel

Eurecom – Engin Kirda

Technical University of Vienna – Paolo Milani Comparetti

Performed under EU financing

(WOMBAT, SysSec, i-Code projects)