OS AGNOSTIC MALWARE DETECTION AND MODELING

OUR AGENDA

1 Statistiche

2 Hypervisor

3 Parser

4 Detection 1

5 Detection 2

MMAS

Progetto sperimentale di tesi di Laurea Magistrale

Supervisione: Roberto Giacobazzi, Vivek Notani

Realizzazione: Stefano Maistri

CHI SONO

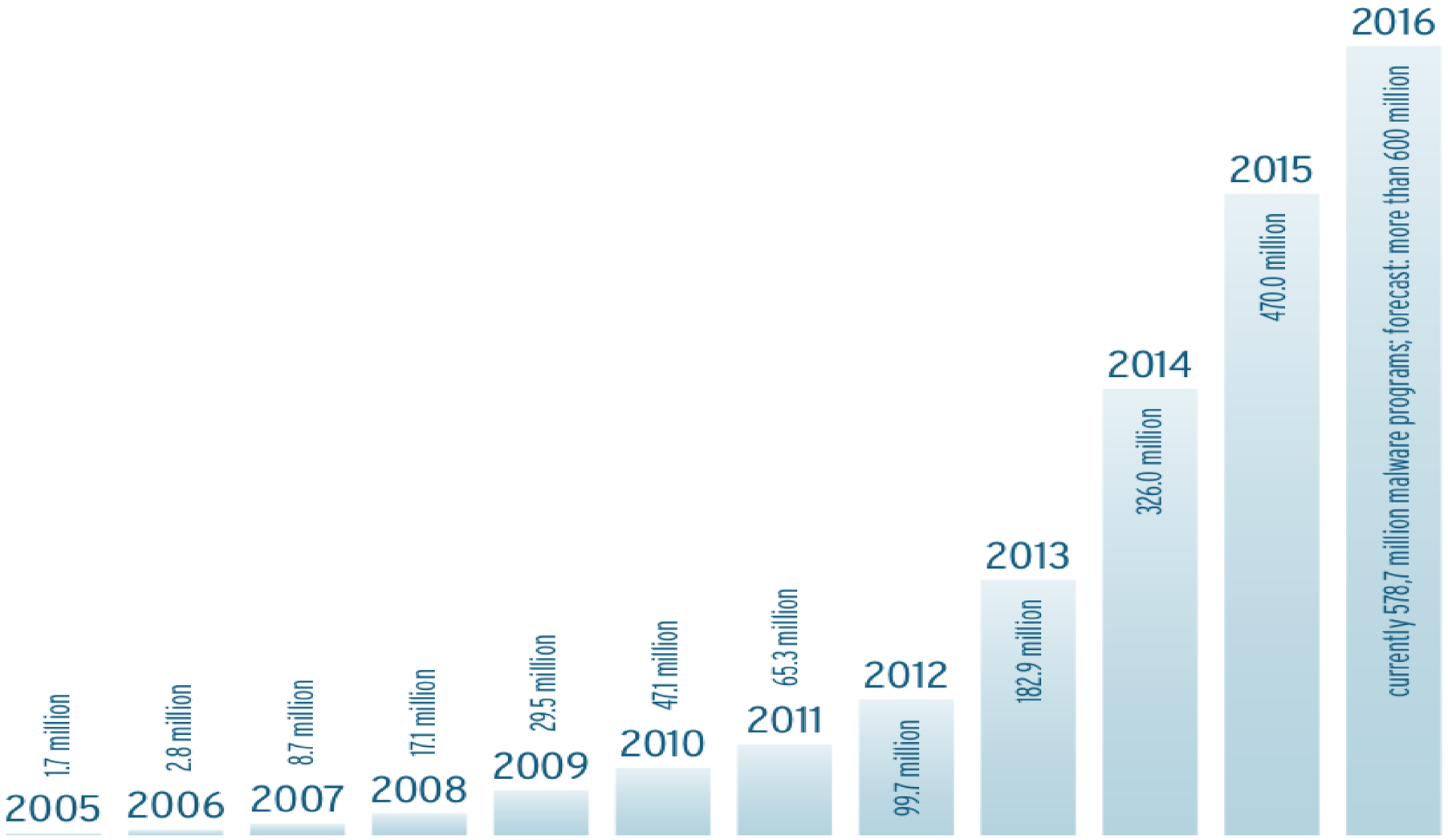
Laurea in Informatica Generale presso l'Università degli Studi di Verona,
Facoltà di Scienze MM. FF. NN. (2013/2014)

Laurea Magistrale in Ingegneria del Software e Sicurezza presso l'Università degli Studi di Verona,
Facoltà di Scienze MM. FF. NN. (18/10/2017)

Security Consultant presso Minded Security (luglio 2017)

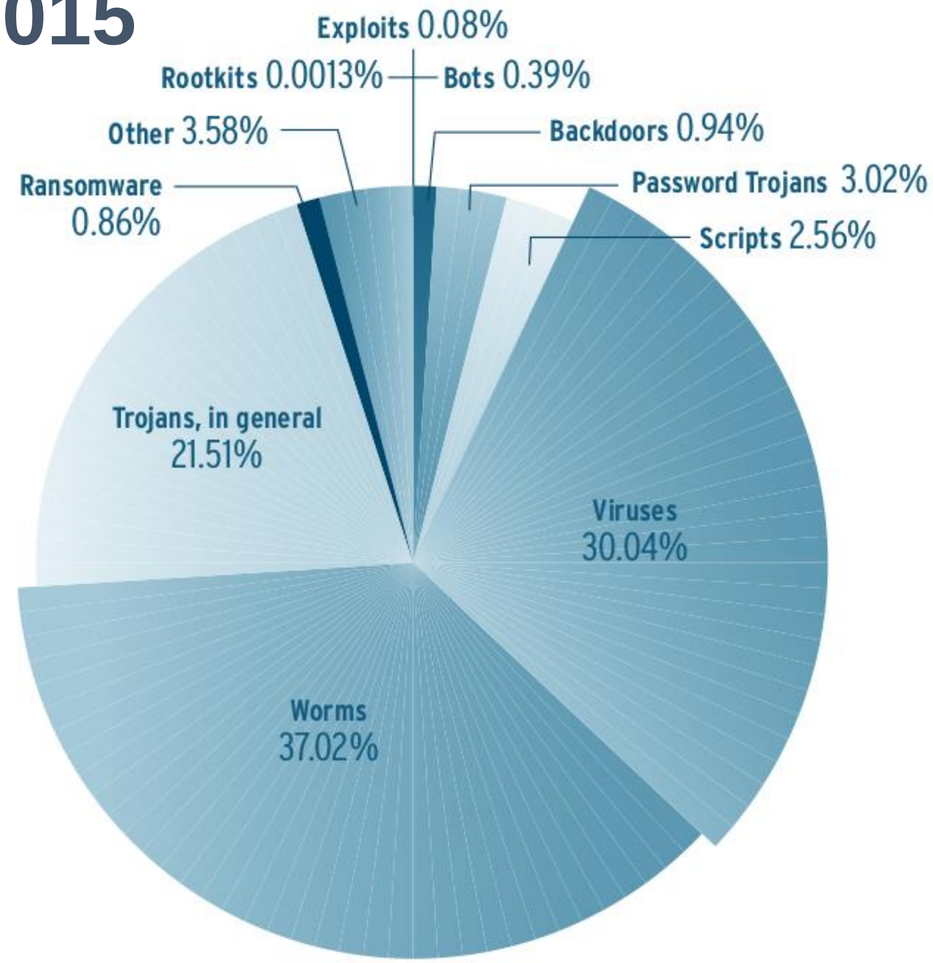
STATISTICHE

STATISTICHE

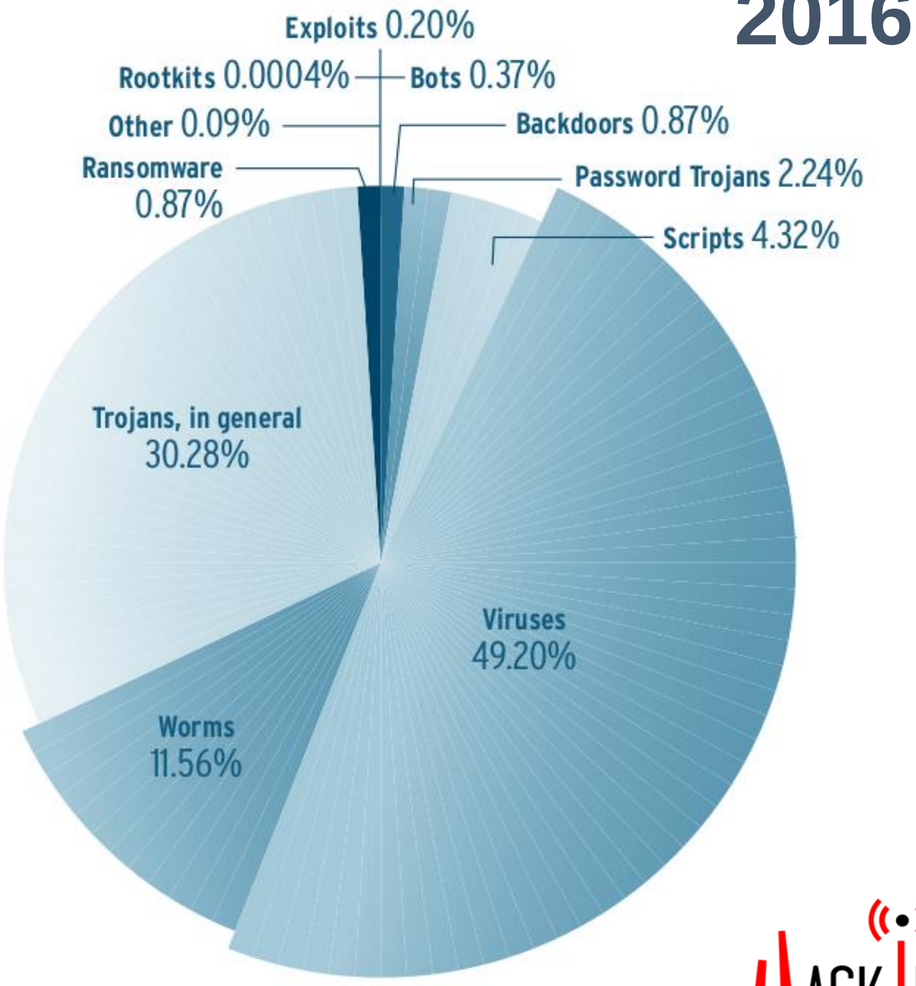


WINDOWS

2015

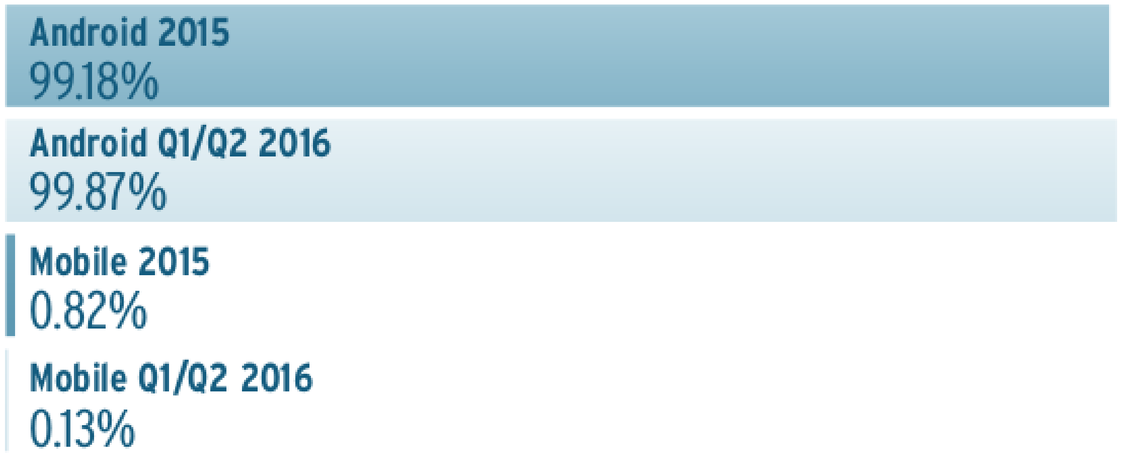


2016

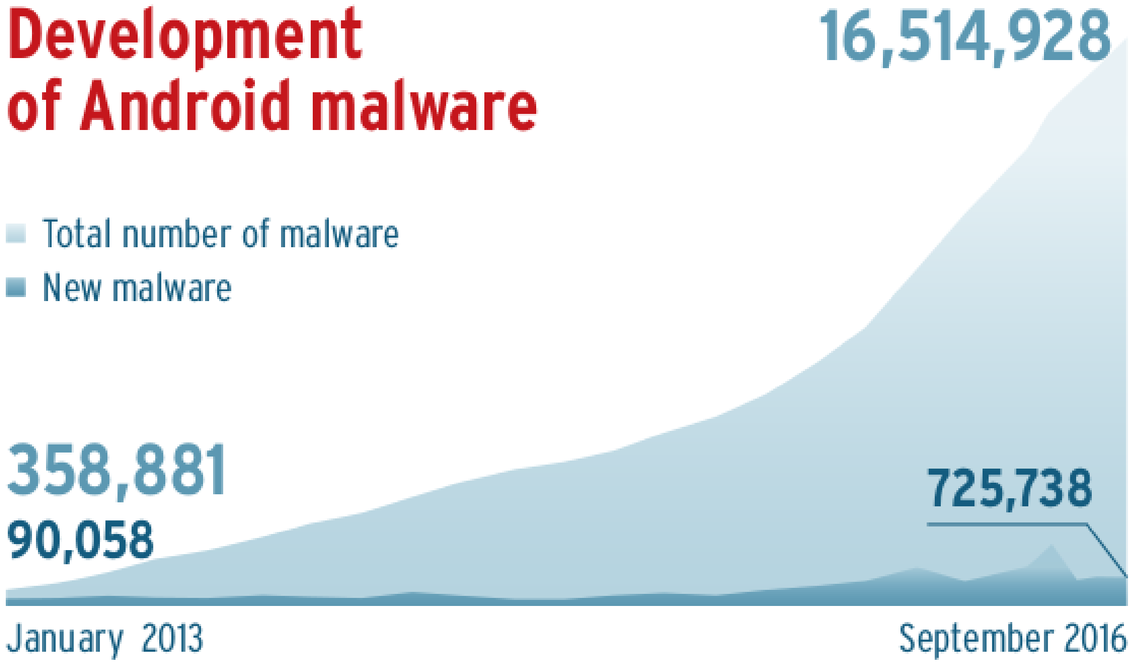


ANDROID

Android malware vs. mobile

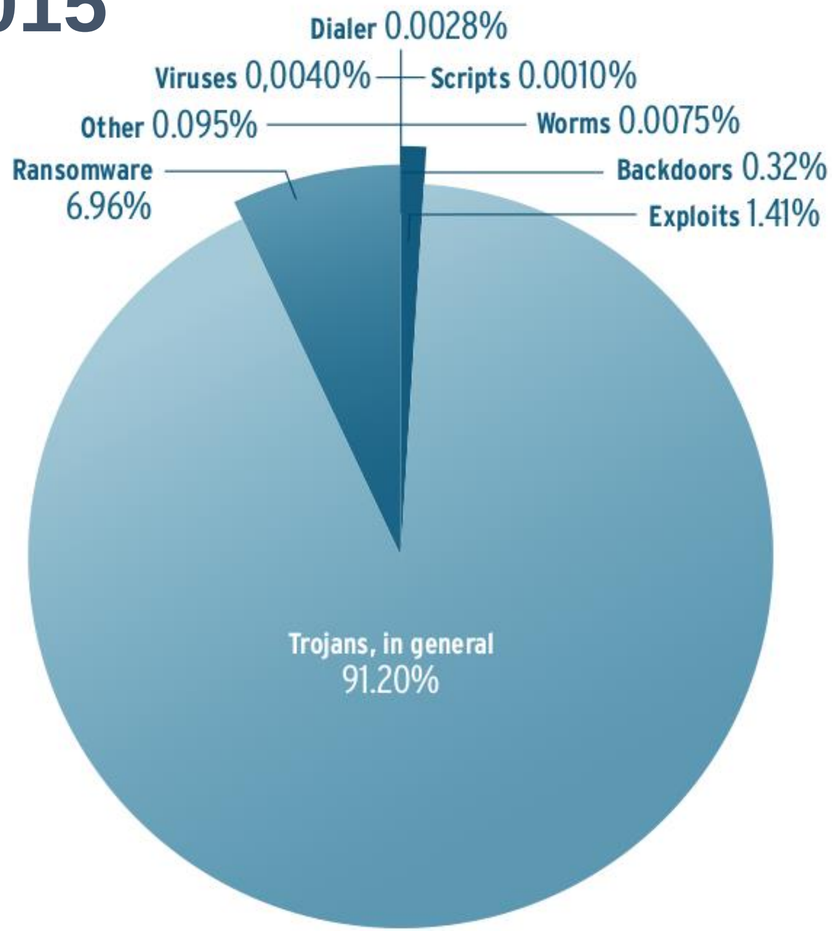


Development of Android malware

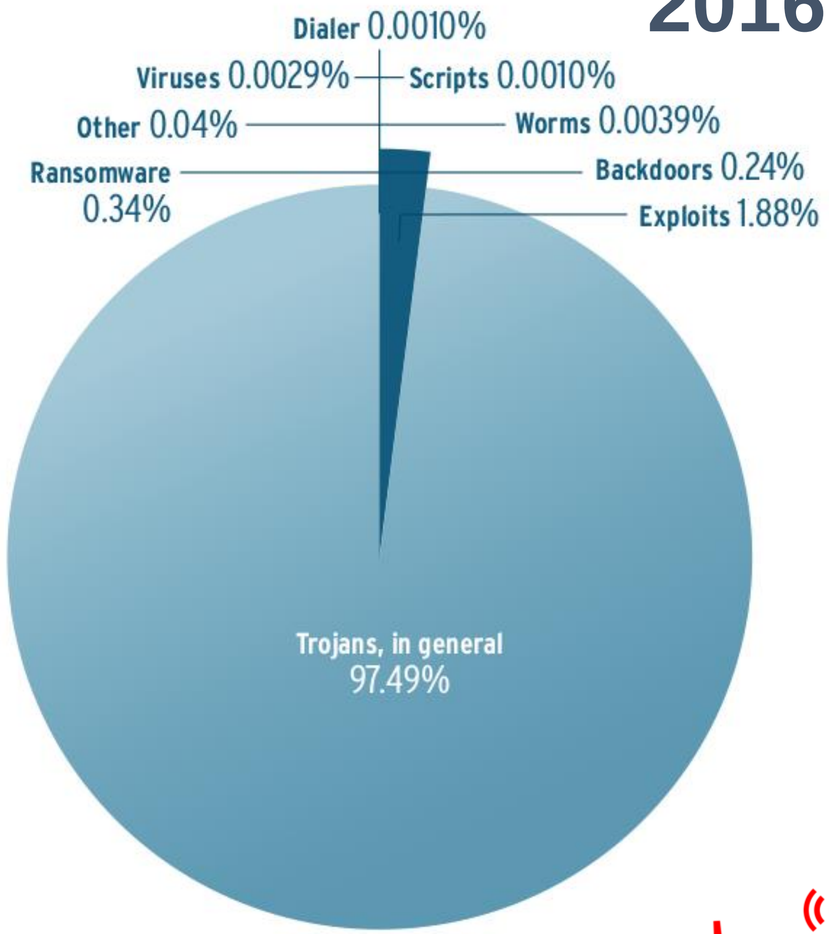


ANDROID

2015



2016



IDEA

- **Cambia** la struttura
- **Restano invariate**: lo scopo finale e le istruzioni

- **Astrarre** la detection dal sistema operativo in uso

- **Modellare** specifici comportamenti malevoli

COMPONENTI MMAS

HYPERVISOR INTROSPECTION

OS AGNOSTIC PARSER

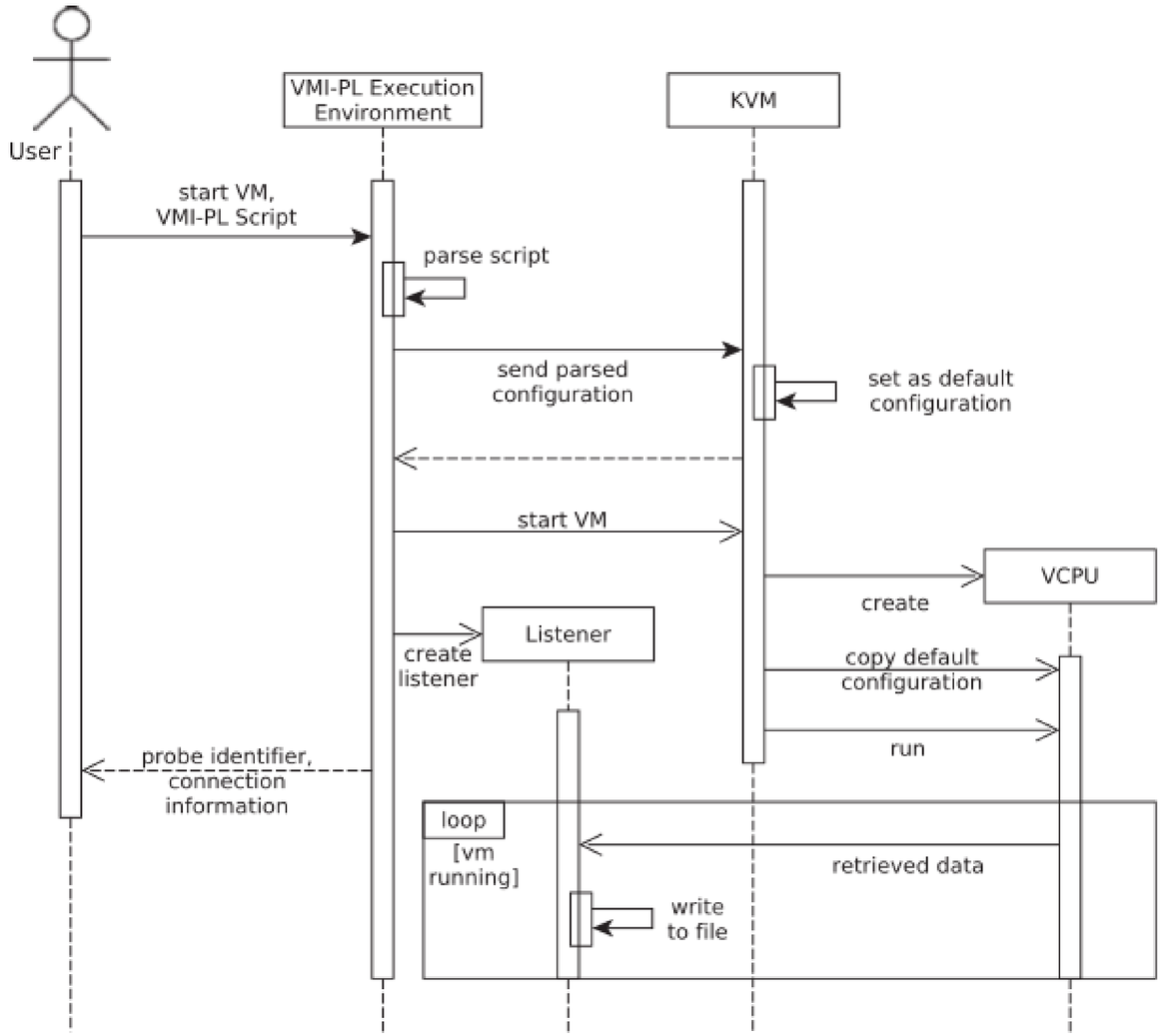
MODELING AND DETECTION

HYPERVISOR INTROSPECTION

API & SYSTEM CALLS

1. Memorizzare tutte le API e le System call richieste dal malware
2. Memorizzare tutte le risorse richieste e modificate dal malware
3. Memorizzare l’interazione del malware con l’utente o con il server di C&C

VMI-PL



- Data Probes
- Event Probes
- Stream Probes

- OS - Hardware Level

- Istruzioni per la riconfigurazione

VMI-PL

ESEMPI

1 Tracciare ciclo di vita di un programma

```
CRWrite(3){
        ReadRegister(CR3, /tmp/schedules_processes)
}
ExecuteAt(0xc104f060){
        ReadRegister(CR3, /tmp/terminated_processes)
}
```

VMI-PL

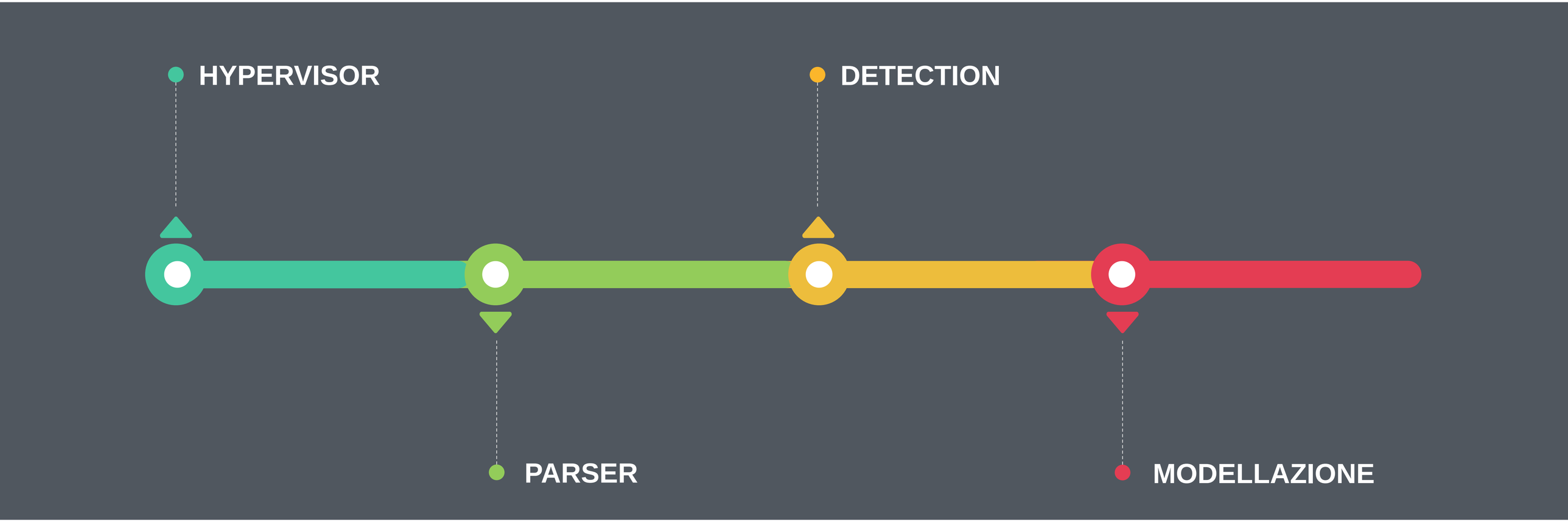
ESEMPI

2 Generare un log delle System Call

```
Configuration{
        SyscallDispatcherAddress: 0xc15780e8
        SyscallNumberLocation: EAX
        SyscallInterruptNumber: 128
}
Syscall{
        ReadRegister(EAX, #syscall_monitor)
        ReadRegister(CR3, #syscall_monitor)
}
```

OS AGNOSTIC PARSER

FLUSSO DI API & SYSCALL



TRADUZIONE


```
OS1: A() OS2: B() OS3: C() OS4: D() ABCD()
```

INCOMPATIBILITY

1. No corrispondenza tra i parametri delle funzioni
2. No corrispondenza tra i valori di ritorno
3. No corrispondenza diretta tra le funzioni
4. No corrispondenza univoca tra le funzioni

INCOMPATIBILITY

1. No corrispondenza tra i parametri delle funzioni

```
int open(
    const char *pathname,
    int flags,
    mode_t mode
);
```

```
HFILE WINAPI OpenFile(
  _In_  LPCSTR     lpFileName,
  _Out_ LPOFSTRUCT lpReOpenBuff,
  _In_  UINT       uStyle
);
```

- `Linux: open(const char *p, int f, mode_t m)`
- `Windows: OpenFile(LPCSTR l, LPOFSTRUCT lo, UINT U)`
- `UniversalOpen(L: const char *p, L: int f, L: mode_t m,`
  `W: LPCSTR l, W: LPOFSTRUCT lo, W: UINT U)`

INCOMPATIBILITY

2. No corrispondenza tra i valori di ritorno

```
int open(
    const char *pathname,
    int flags,
    mode_t mode
);
```

```
HFILE WINAPI OpenFile(
  _In_  LPCSTR     lpFileName,
  _Out_ LPOFSTRUCT lpReOpenBuff,
  _In_  UINT       uStyle
);
```

Linux: {int} Windows: {HFILE WINAPI} UniversalOpen {L: int, W: HFILE WINAPI}

INCOMPATIBILITY

3. No corrispondenza diretta tra le funzioni

```
HRESULT WINAPI WerReportAddDump(
  _In_     HREPORT                          hReportHandle,
  _In_     HANDLE                           hProcess,
  _In_opt_ HANDLE                           hThread,
  _In_     WER_DUMP_TYPE                    dumpType,
  _In_opt_ PWER_EXCEPTION_INFORMATION       pExceptionParam,
  _In_opt_ PWER_DUMP_CUSTOM_OPTIONS         pDumpCustomOptions,
  _In_     DWORD                            dwFlags
);
```

Windows: funzione(parametri) GlobalFunzione(W: parametri)

INCOMPATIBILITY

4. No corrispondenza univoca tra le funzioni

1	2	5
OS 1	**OS 2**	**UNIVERSALE**
SYSCALL A()	SYSCALL B() SYSCALL C() SYSCALL D()	SYSCALL ABCD()

```
OS1: A()  OS2: [B()  C()  D()]  ABCD()
```

COME MAI OS AGNOSTIC

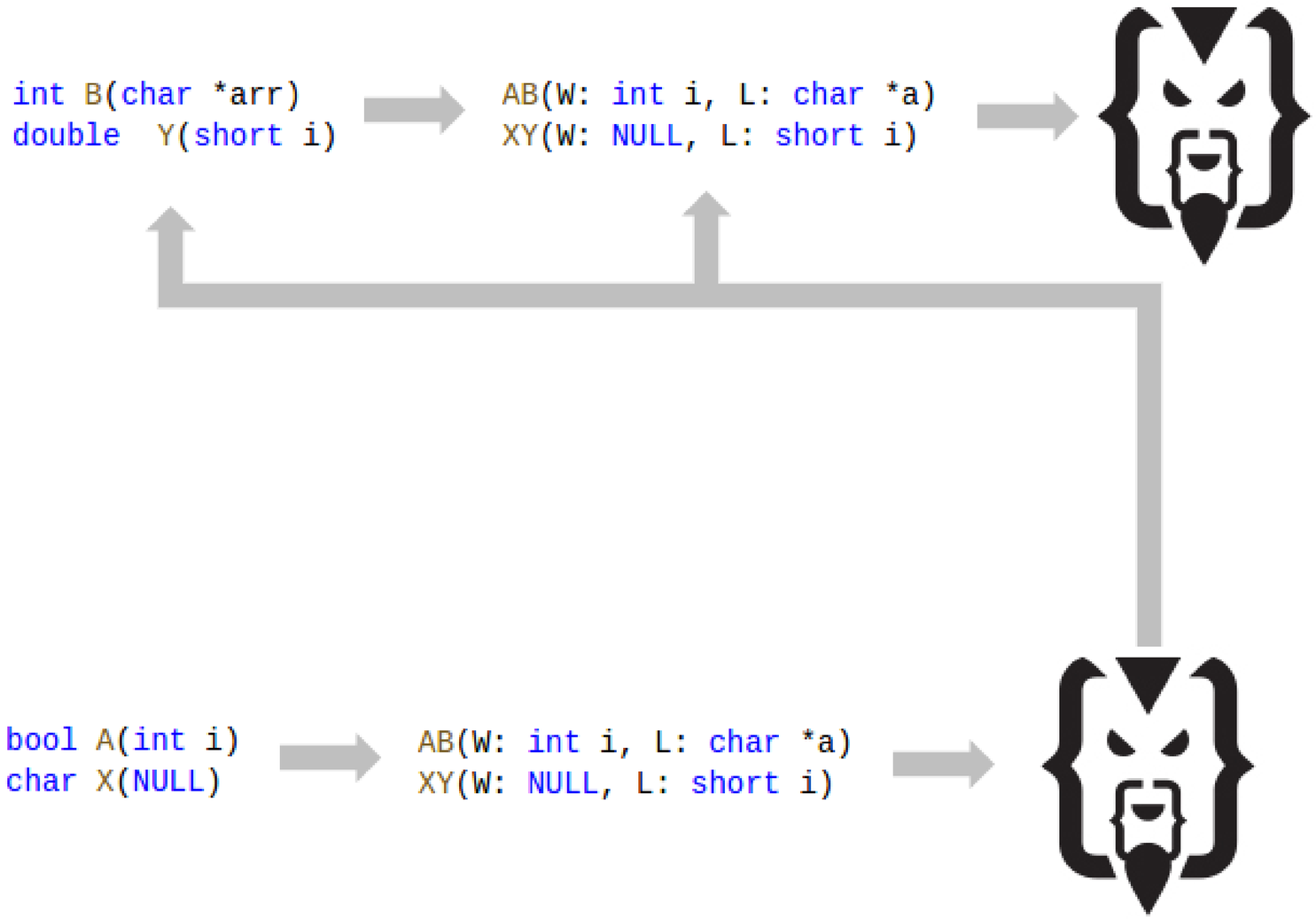
```
int B(char *arr)
double  Y(short i)
```

```
AB(W: int i, L: char *a)
XY(W: NULL, L: short i)
```

```
bool A(int i)
char X(NULL)
```

```
AB(W: int i, L: char *a)
XY(W: NULL, L: short i)
```

COME MAI OS AGNOSTIC



MODELING AND DETECTION

MODELLI

ALTO LIVELLO

- OS Agnostic
- Definito manualmente
- Template d’attacco

BASSO LIVELLO

- OS Specific
- Definito automaticamente
- Implementazione

DETECTION E RICERCA

DETECTION 1

1.ESTRAZIONE SEQUENZE DI API

2.RICERCA DI MODELLI DI ALTO LIVELLO

DETECTION 2

1.RICERCA DI CORRISPONDENZA TRA MODELLI

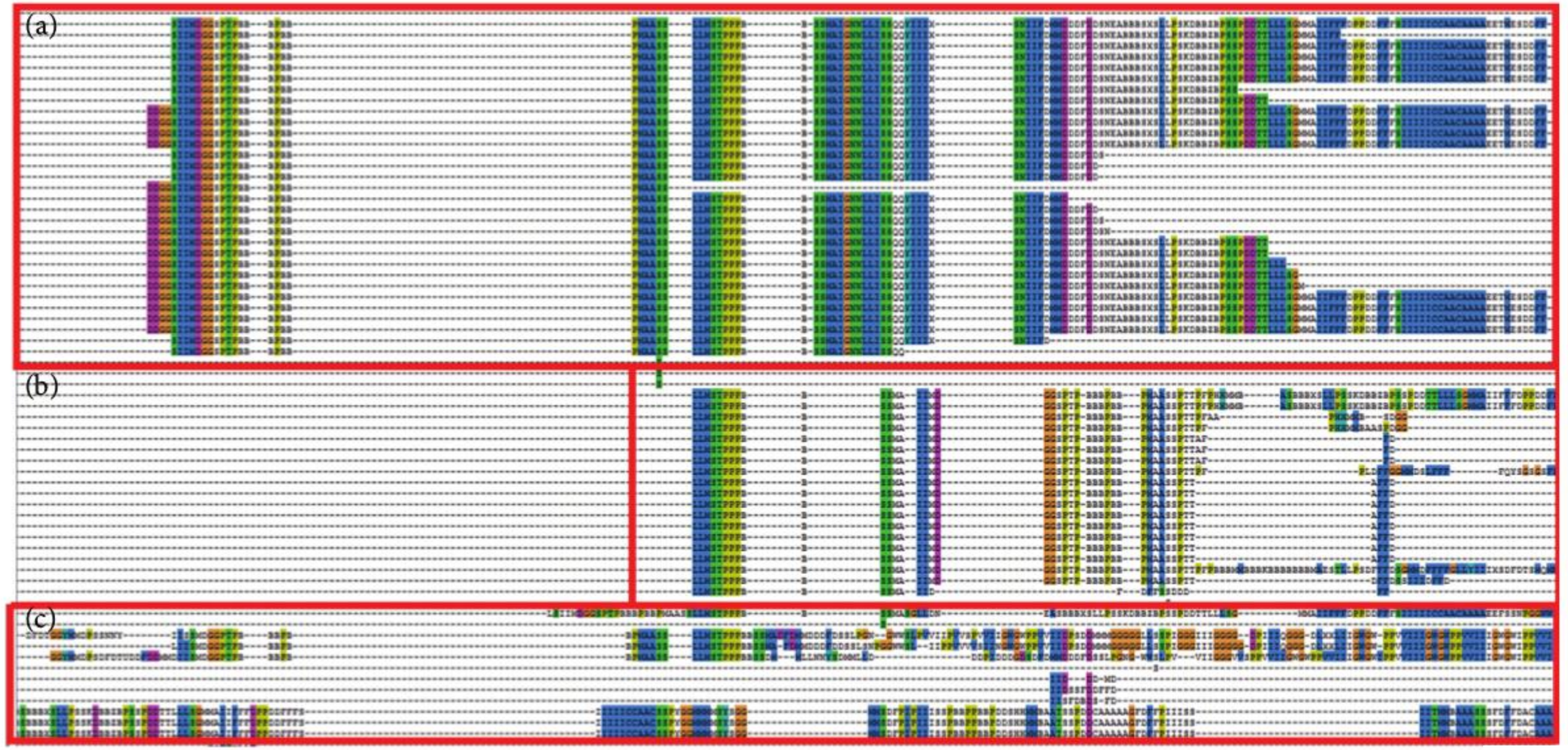
2.GENERAZIONE DI NUOVI MODELLI

3.RICERCA DI COMPONENTI CONDIVISE E FUNZIONALITA’ DORMIENTI

DETECTION 1

1. Definizione manuale di comportamenti malevoli
2. Applicazione algoritmi di MSA sul malware in analisi
3. Calcolo sequenza LCS e ricerca di componenti malevoli

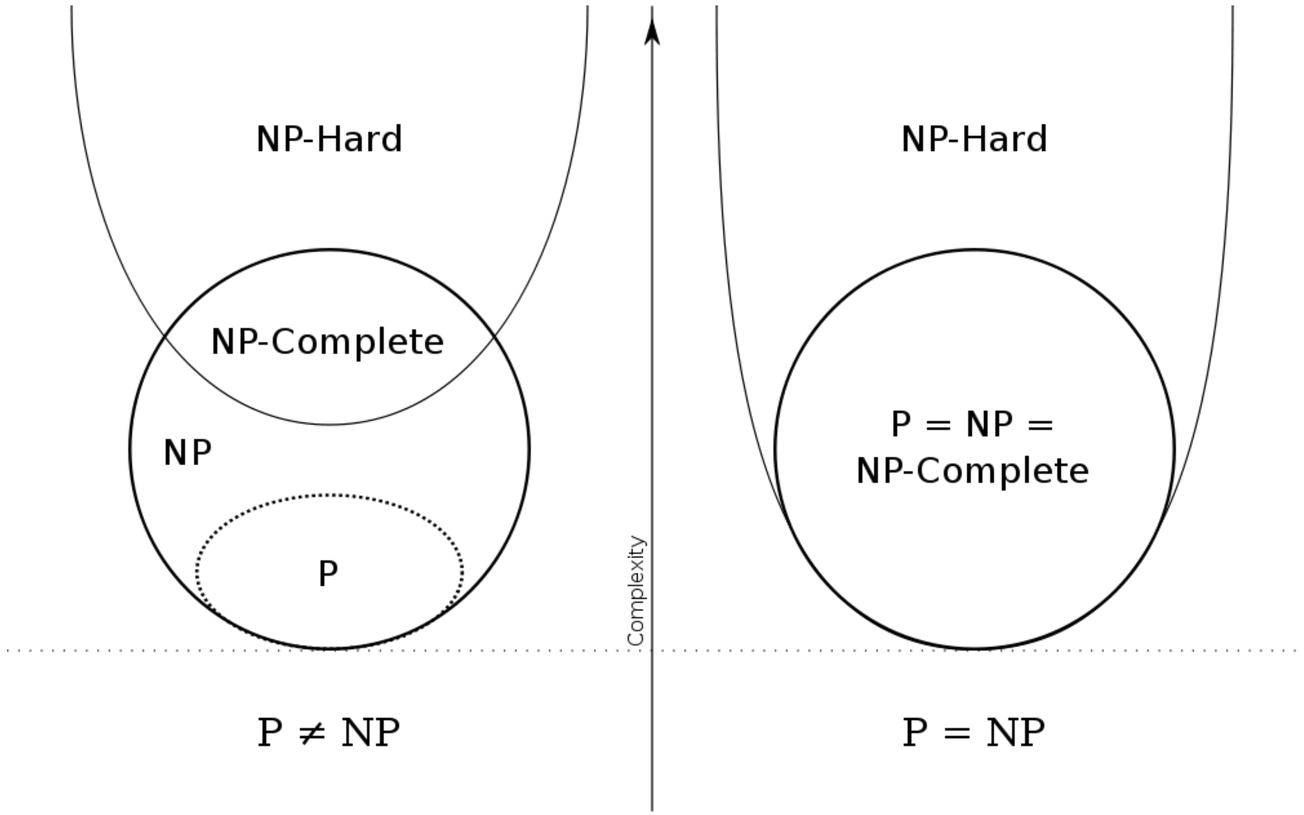
DETECTION 1



DETECTION 1

1. OpenProcess: creato un handler al processo target
2. VirtualAllocateEx: allocato spazio nella memoria del processo target
3. WriteProcessMonitor: scritto il nome della DDL e il path completo al file
4. CreateRemoteThread: indotto il processo target a ricaricare la dll

P = NP ?

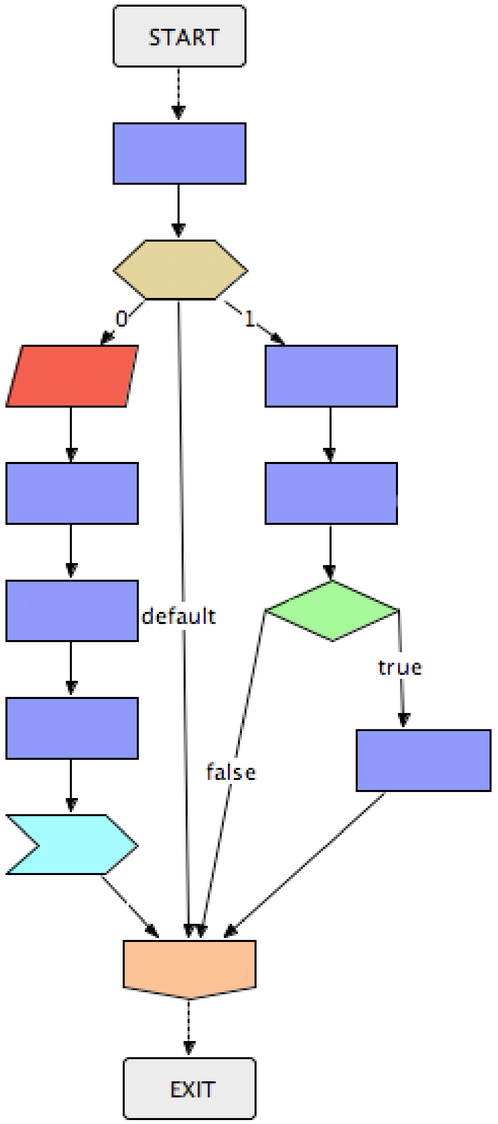


MSA ∈ NP-Complete, LCS ∈ NP-Hard

DETECTION 2

1. Identificazione dell’implementazione del comportamento malevolo
2. Identificazione STATICA di componenti condivise e dormienti
3. Identificazione DINAMICA di componenti condivise e dormienti

OPERAZIONI PRELIMINARI



1 Estrazione CFG del malware

2 Conversione del modello (MALSPEC)

3 Estrazione funzioni interessanti Rb

IDENTIFICARE L'IMPLEMENTAZIONE

PROGRAM SLICING

IDEA: Se x ∈ Rb allora tutto il codice che prepara i dati di x ∈ Rb

Forward slice: Taint analysis data flow da return → funzione

Backward slice: Log analysis origine dei parametri

IDENTIFICARE L'IMPLEMENTAZIONE

FILTERING

IDEA: Eliminare funzioni general purpose

Filtro White List: X ≠ modello in analisi → X è in white list

Filtro Tainted Data Flow: X non lavora solo con lo slice → X eliminato

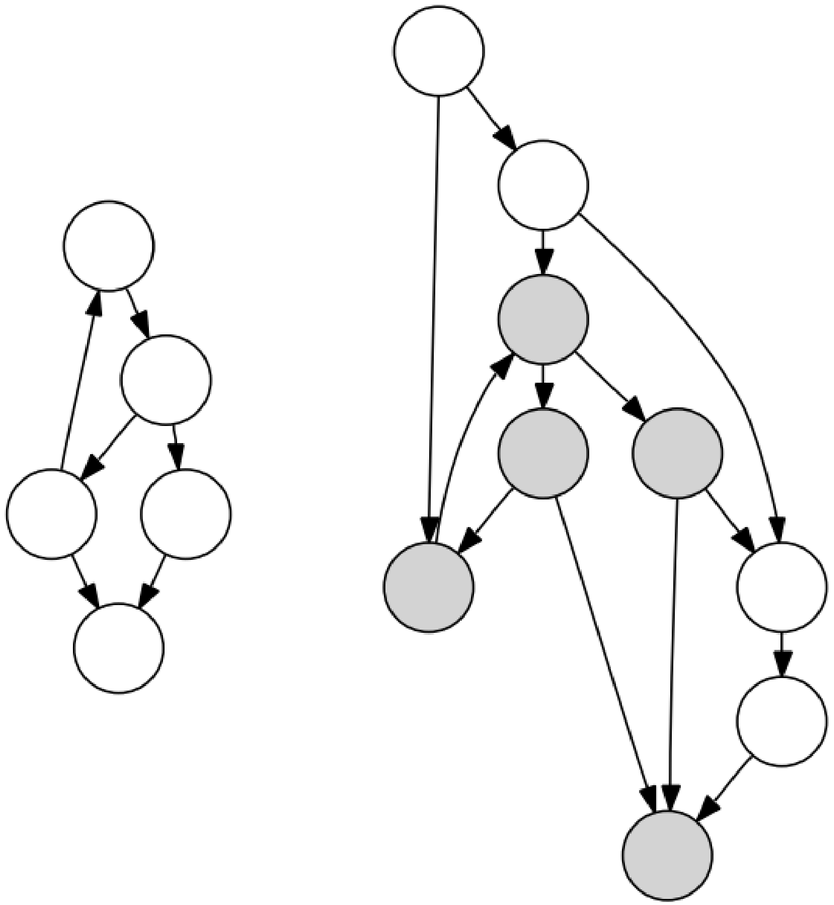
IDENTIFICARE L'IMPLEMENTAZIONE

ESPANSIONE

Filtraggio e slicing estremamente selettivi

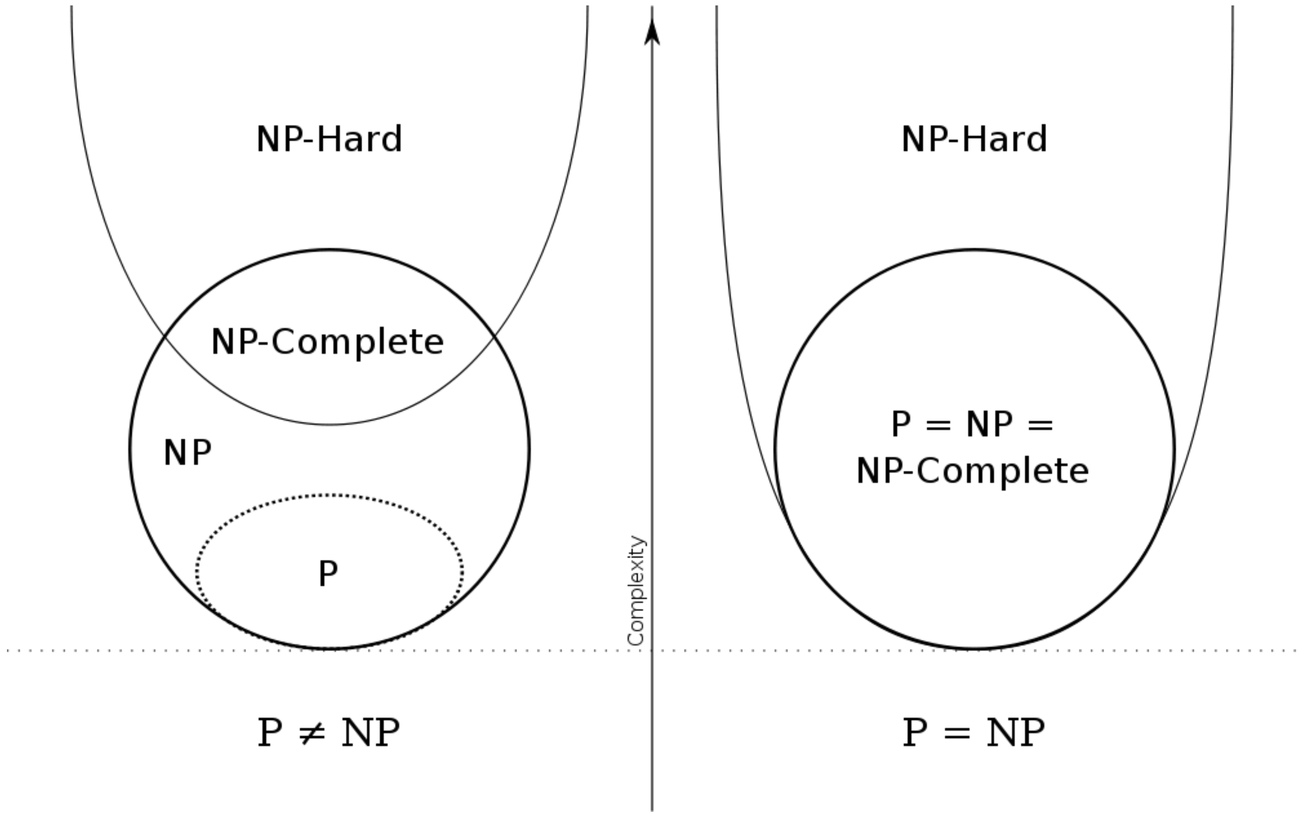
IDEA: X eseguibile sse viene eseguita almeno una operazione dello slice: X ∈ slice

INDIVIDUAZIONE STATICA



1 | Ricerca isomorfismo tra sottografi

P = NP ?



Isomorfismo ∈ NP-Complete

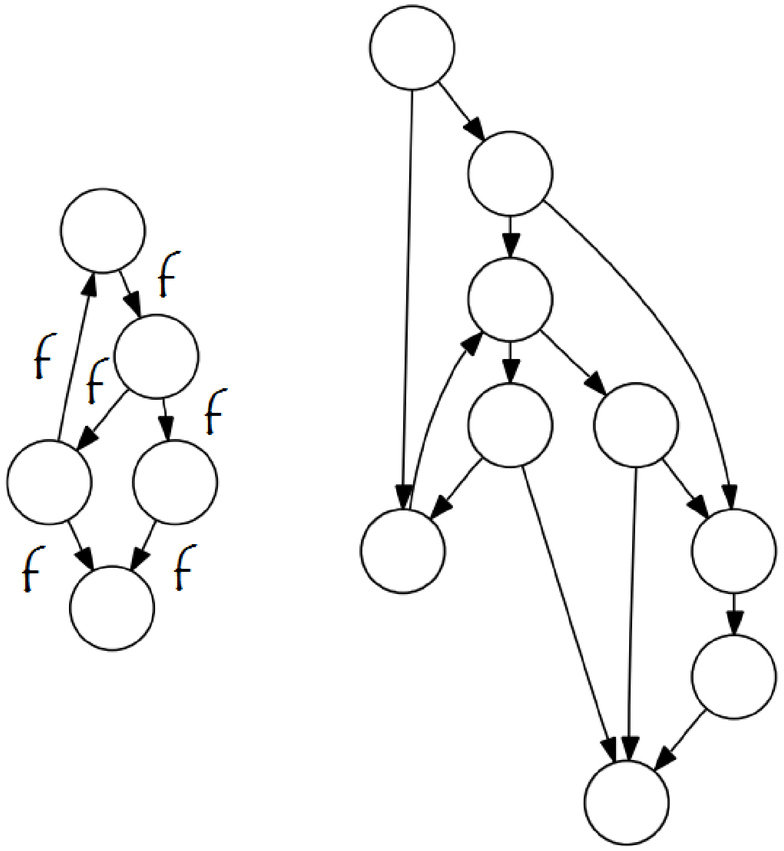
INDIVIDUAZIONE STATICA

1a | Approssimati tutti i possibili sottografi e normalizzati tutti i CFG

1b | Ogni grafo normalizzato viene trasformato in fingerprint tramite hashing

1c | Ricerca di corrispondenza attraverso il confronto tra stringhe

INDIVIDUAZIONE DINAMICA



1 | Calcolata espressione simbolica per ogni input parameter

2 | Conversione del modello (da OS Agnostic a Simbolic)

3 | Engine: Kolbitsch e Comparetti

CONTATTI

Phone

+39 339 395 0085

Email

stefano.maistri@mindedsecurity.com

stefanomaistri1990@gmail.com

stefano.maistri@studenti.univr.it

Social Media

Twitter: @SMaistri

Linkedin: Stefano Maistri