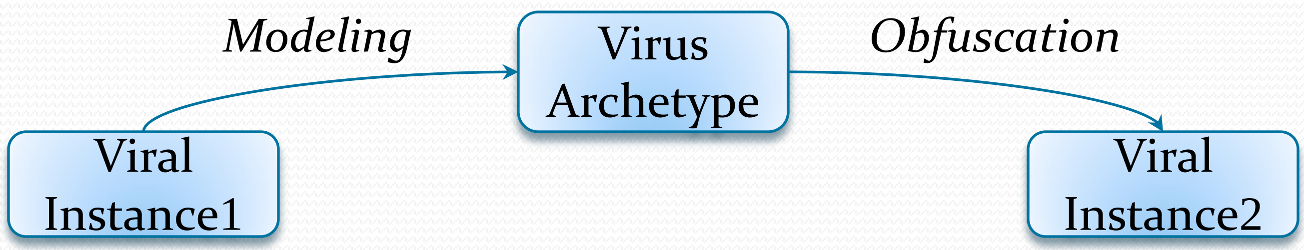
Obfuscation Techniques for Metamorphic Viruses

Borello Jean-Marie
jean-marie.borello@dga.defense.gouv.fr
Centre d'ELectronique de l'ARmement (CELAR)

Mé Ludovic
lme@rennes.supelec.fr
Supelec, équipe SSIR

Introduction

- **Definition** : a metamorphic virus is a program able to model itself in order to replicate.



- **Properties** :
  - Each instance looks different but have the same behavior.
  - Low level pattern matching is impossible.
- **Main assumption** : as a metamorphic virus can model itself, another program could do so.

Plan

- State of art of metamorphic viruses.
  - Obfuscation.
  - Detection.
- Limits in metamorphic viruses detection.
  - Formal impossibility of a perfect detection.
  - Difficulty of a reliable static detection.
- Approach of obfuscation.

Obfuscation Techniques

- **Definition** : Informally speaking obfuscation stands for the process of making a piece of code as difficult to understand as possible.

- Obfuscation works at two levels :
  - Data flow level.
  - Control flow level.

Instructions Substitution

- Exchange two instructions sequences which have the same semantics.

Simple Instructions		Sequence of Instructions	
XOR	Reg,Reg	MOV	Reg,0
MOV	Reg,Imm	PUSH POP	Imm Reg
OP	Reg1,Reg2	MOV OP MOV	Mem,Reg1 Mem,Reg2 Reg1,Mem

Instructions Permutation

- Only the instructions order is changed

Simple Instructions	Sequence of Instructions
MOV ecx,104h MOV edi,dword ptr [ebp+08h] MOV esi,dword ptr [ebp+0Ch] REPNZ MOVSB	MOV edi,dword ptr [ebp+08h] MOV ecx,104h MOV esi,dword ptr [ebp+0Ch] REPNZ MOVSB

Dead code insertion

- Insertion of useless code.

Dead codes	Meanings
`ADD Reg,0`	Reg ← Reg+0
`MOV Reg,Reg`	Reg ← Reg
`OR Reg,0`	Reg ← Reg|0
`AND Reg, -1`	Reg ← Reg&0FFFFFFFFh

Variable substitution

- Change only the variable assignments.

First Instance	Second Instance
`POP edx`	`POP eax`
`MOV edi,04h`	`MOV ebx,04h`
`MOV esi,ebp`	`MOV edx,ebp`
`MOV eax,0Ch`	`MOV edi,0Ch`
`ADD edx,088h`	`ADD eax,088h`

Control Flow Alteration

- Change a program control flow by inserting some conditional and unconditional branches.

Original Program	First alteration	Second alteration
Instruction1	Instruction4	Start:
Instruction2	Instruction5	Instruction1
Instruction3	Jump	Jump
Instruction4	Start:	Instruction3
Instruction5	Instruction1	Instruction4
Instruction6	Instruction2	Jump
	Jump	Instruction2
	Garbage	Jump
	Instruction3	Instruction5
	Jump	Instruction6
	Instruction6	

Metamorphic Viruses Detection

- Static detection based on low level pattern matching
- Main assumption :

  as a metamorphic virus is able to model itself in order to replicate, another program should be able to do so.
- Main idea : use high level patterns

Metamorphic Viruses Detection

- High Level Pattern = optimized Control Flow Graph (CFG)
  - Build the CFG
  - Optimize the Data Flow Graph (DFG)
  - Optimize the CFG

Metamorphic Viruses Detection

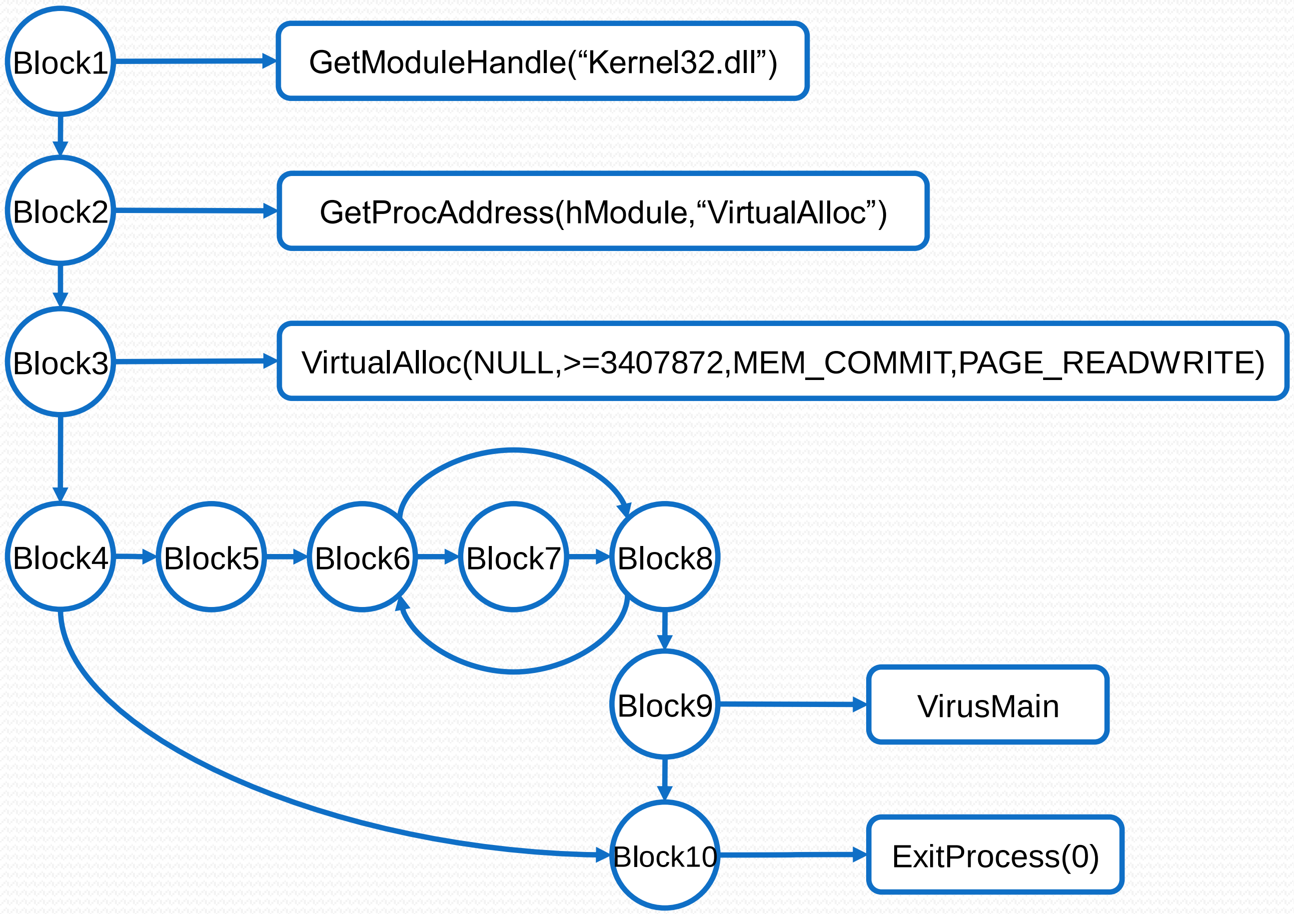
- Meta-representation
  - Assignment =
  - Call to a procedure CALL
  - Return of a procedure RET
  - Conditional branch JCOND
  - Unconditional branch GOTO

- Optimizations:
  - Data propagation
  - Dead code elimination
  - Algebraic simplifications
  - Control Flow Graph structuration (Loops,…)

Metamorphic Viruses Detection

Original code	Meta-representation	After optimization
01. MOV esi, esi	esi=esi	
02. MOV dword_A, 0	dword_A=0	dword_A=0
03. MOV esi, dword_A	esi=dword_A	esi=0
04. PUSH esi	esp=esp-4	[esp-4]=0
05.	[esp]=esi	
06. MOV dword_B,offset ExitProcess	dword_B=&ExitProcess	dword_B=&ExitProcess
07. MOV ebx, dword_B	ebx=dword_B	ebx=&ExitProcess
08. PUSH dword ptr [ebx+0]	esp=esp-4	
09.	[esp]=[ebx]	[esp-8]=ExitProcess
10. POP dword_C	dword_C=[esp]	dword_C=ExitProcess
11.	esp=esp+4	
12. CALL dword_C	CALL dword_C	CALL ExitProcess

Metamorphic Viruses Detection



Limits in reliable static detection

- **Notation** : 2 programs A and B with inputs D_A and D_B are said to be functionally equivalent ($A \equiv B$) iff,
  - $D_A = D_B$
  - $\exists x \in D_A$, $A(x)=B(x)$
- **Definition 1**: a program D_V reliably detects a metamorphic virus V iff for all program P,
  - $D_V(P)$ returns "true" if $P \equiv V$
  - $D_V(P)$ returns "false" else
- **Proposition 1**: no algorithm can claim if for all programs A and B, $A \equiv B$.

Limits in reliable static detection

- **Corollary 1** : detecting a metamorphic virus as defined in definition 1 is an undecidable problem.

- **Proposition 2** : in the assumption that all paths are potentially executable, for all programs A and B such that, $\forall x \in D_A$, $A(x) \neq \perp$ determining if $B \equiv A$ is a $\mathcal{NP}$-hard problem.

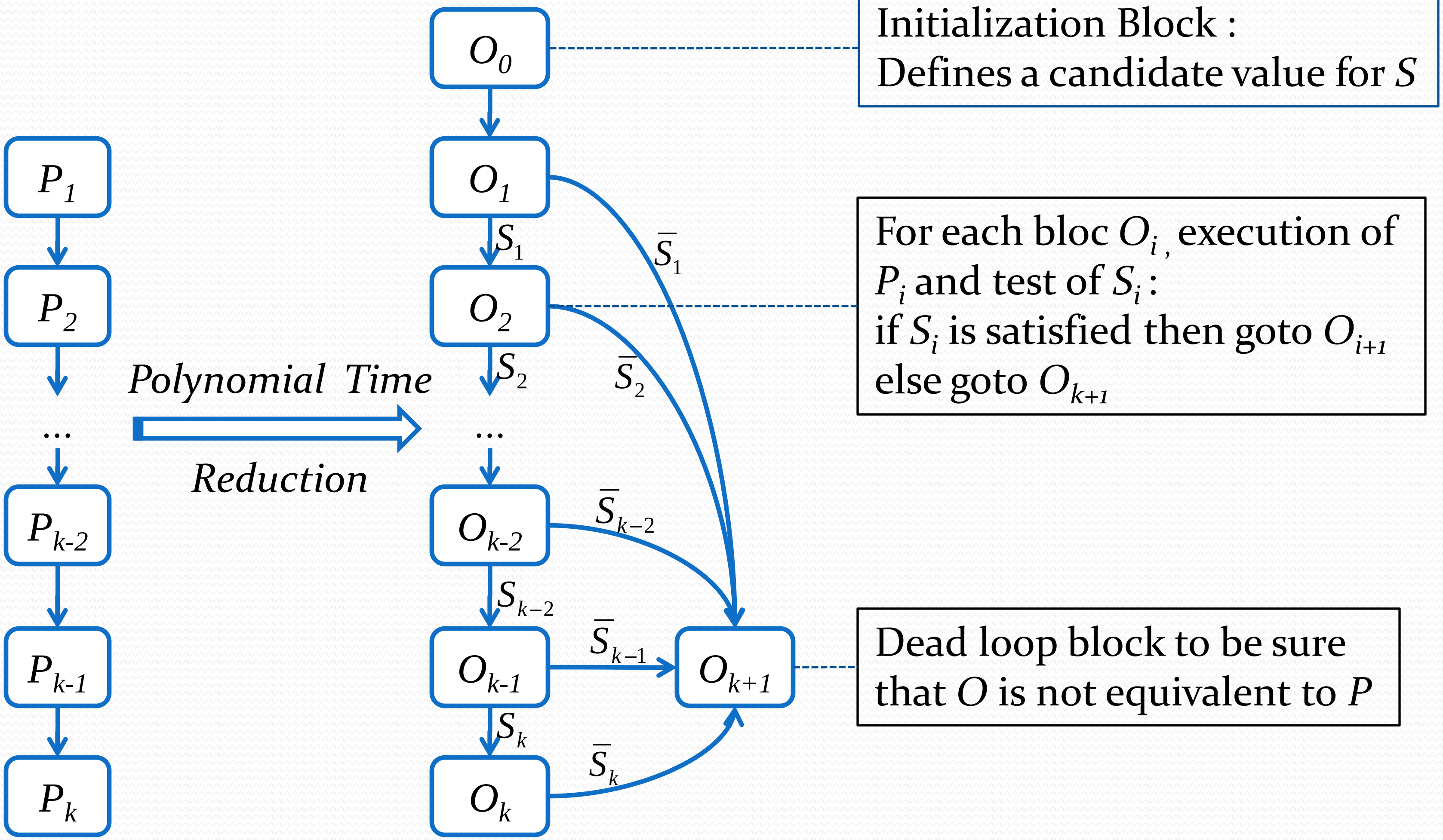
Limits in reliable static detection

- **Sketch of proof** :
  - We consider and instance S of the satisfiability problem known to be $\mathcal{NP}$ - complete. Then we build in polynomial time a program O from a program P such that, S satisfiable iff there exists a path in O such that $O \equiv P$

$$S = \bigcap_{i=1}^{n} \left(l_{i,1} \vee l_{i,2} \vee l_{i,3} \right)$$

    - $\{v_1, v_2, \ldots, v_n\}$ is a set of Boolean variables
    - $\forall (i, j) \in [1, n] \times [1,3], \exists k \in [1, m], l_{ij} = v_k$ or $l_{ij} = \overline{v_k}$
  - We split the set $[1, n]$ into consecutives elements denoted $(u_i)_{i \in [1,k]}$
    - $S = \bigcap_{i=1}^{k} S_i$ with $S_i = \bigcap_{j=\min(u_i)}^{\max(u_i)} \left(l_{i,1} \vee l_{i,2} \vee l_{i,3} \right)$

Limits in reliable static detection



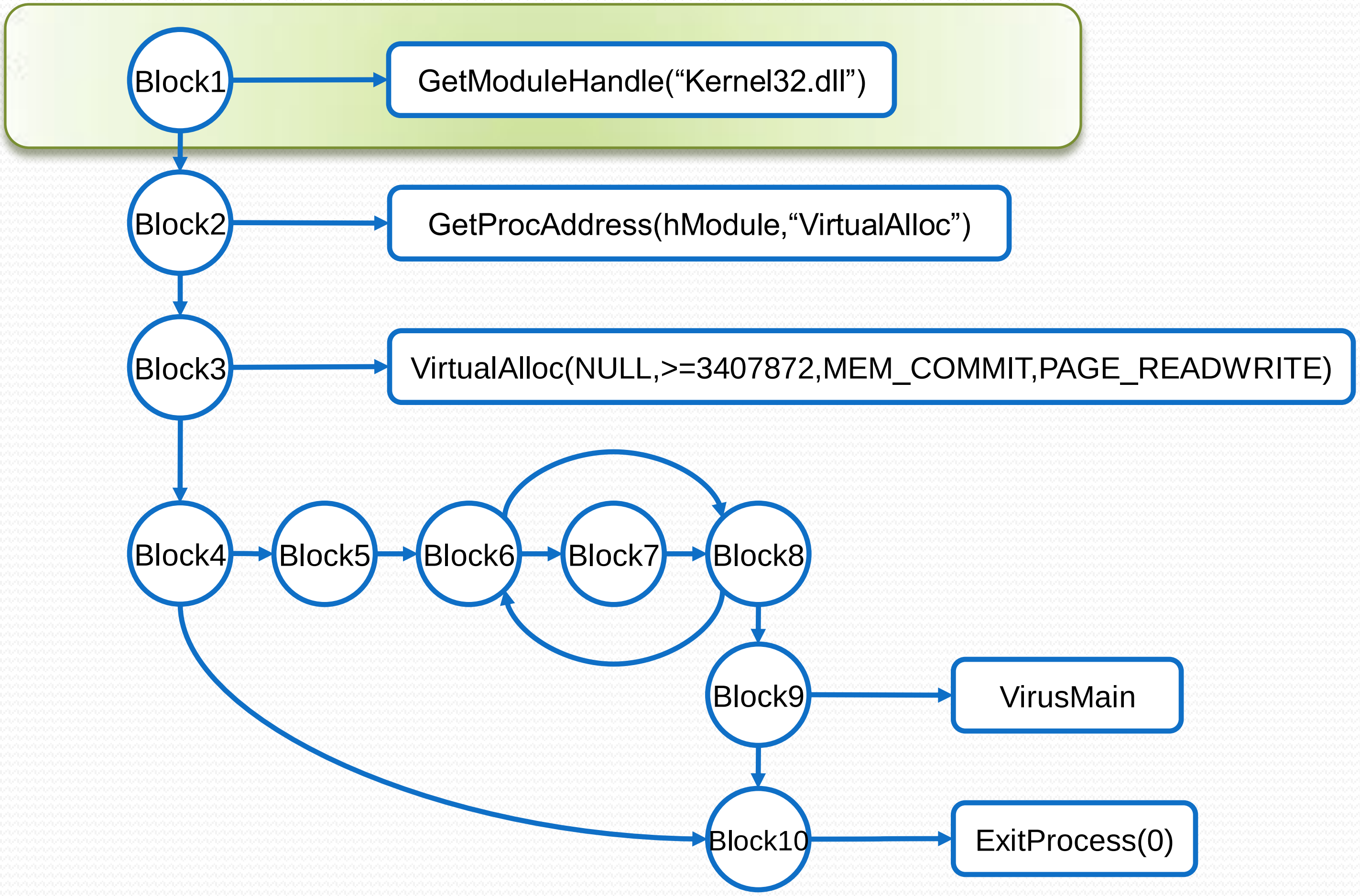
Limits in reliable static detection

- **Corollary 2** : detecting a metamorphic virus as assumed in proposition 2 is a $\mathcal{NP}$ - hard problem.

- This result is a generalization of Spinellis one about the difficulty of polymorphic viruses detection.

- **Consequences** :
  - Only approximate detection techniques are computable.
  - Advanced obfuscation techniques based on control flow modification can make static analysis very difficult.

Obfuscation Approach



Obfuscation Approach

- Randomly split a program P in k lumps.

Obfuscation Approach

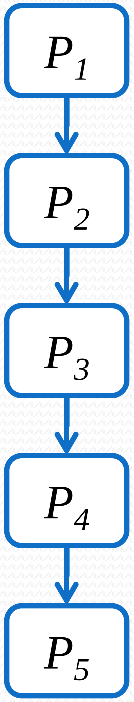
P_1	MOV ebp, 7ABBEDE5h
	MOV dword_40C89E, 0B7777E5Fh
	AND ebp, dword_40C89

P_2	MOV dword_40C2F4, ‘NrEK’
	MOV eax, dword_40C2F4
	LEA ebx, ds: ‘llD.’
	LEA edx, [ebx]
	LEA edi,[edx+0]

P_3	MOV dword_40C0B4, edi
	MOV dword_40C0B0, ebp
	MOV dword_40C0AC, eax
	LEA edi, large ds:0
	MOV dword_40C0B8, edi
	PUSH offset dword_40C0AC
	POP dword_40C6C4
	MOV ecx, dword_40C6C4

P_4	MOV dword_40C1F4, ecx
	PUSH dword_40C1F4
	POP dword_40C9BF
	MOV eax, dword_40C9BF
	LEA edi, GetModuleHandleA

P_5	PUSH eax CALL dword ptr[edi]



GetModuleHandle("Kernel32.dll")

Obfuscation Approach

- Randomly split a program *P* in *k* lumps.
- Add some garbage lumps.

G_1	`XOR ebx, ebx`
	`MOV ebp, 24h`
	`MOV eax, 26h`

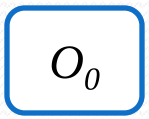
G_2	`MOV ebp, esp`
	`XOR edi, edi`
	`MOV eax, 1F03FFh`

Obfuscation Approach

- Randomly split a program P in k lumps.
- Add some garbage lumps.
- Build the obfuscated program O.

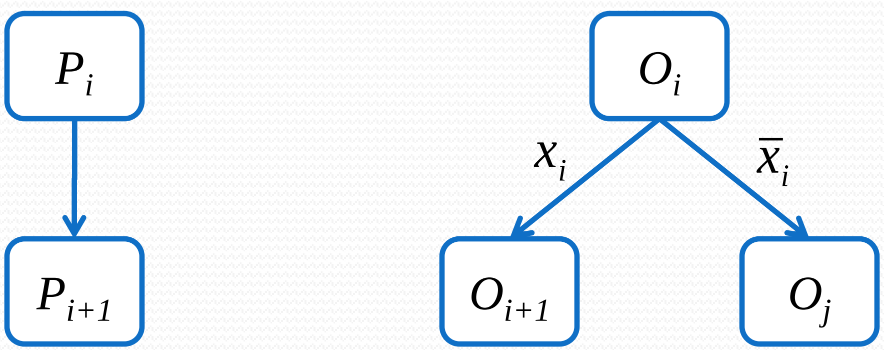
Obfuscation Approach

- First block

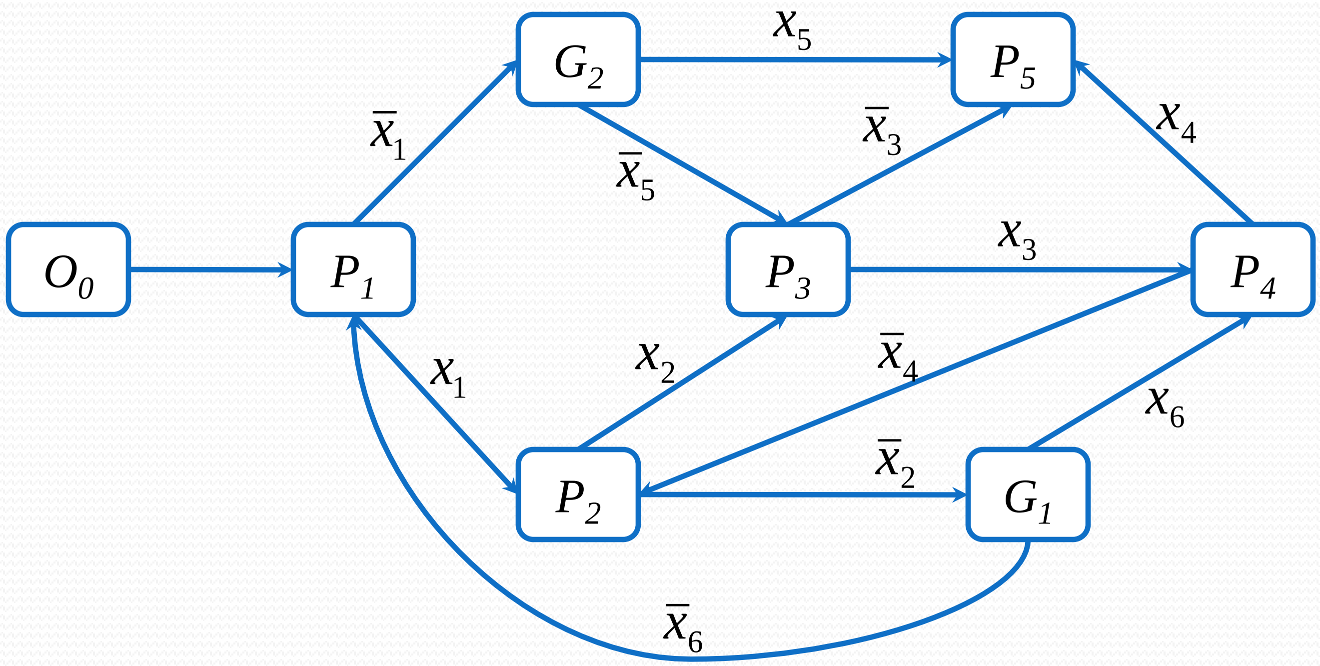


Defines the obfuscation parameter $K=\{x_1,\ldots, x_6\}$

- Build the obfuscated program O



Obfuscation Approach



- K being unknown, we have 2^6=64 executable paths.

Obfuscation Approach

- **Difficulty of determining *K***
  - Mathematics difficulties :
    - Use of Mathematics conjectures like Syracuse one.
    - Difficult Boolean expressions like

      `if (a*(a+1)%2==0)` $\{x_i$`=1;} else {`x_i`=0;}`

  - Dynamic initialization of $K = \{x_1,\ldots,x_m\}$:
    - Use of high level API.
    - $\forall i \in [1,m]$, $\exists j \in [1,m]$, $x_i = H(P_j)$ where H is a Hash function.

Conclusion

- Reliable detection of metamorphic viruses is a $\mathcal{NP}$-hard problem.
- Proposed approach could be used to build metamorphic viruses.
- Should study all the replication cycle.

Thanks for your attention

Questions?