Malware of the Future

When Mathematics work for the Dark Side

Eric Filiol
efiliol@esat.terre.defense.gouv.fr,filiol@esiea-ouest.fr

ESAT Rennes & ESIEA Laval
Laboratoire de virologie et de cryptologie opérationnelles

Hack.lu Conference – October 22nd, 2008

Introduction

Claim (AV industry)

« We detect 100 % of Malware even the unkown ones! »


G DATA
SECURITY
Technologie primée
Deux modules anti-virus pour une double protection
DOUBLESCAN
OUTBREAKSHIELD
AntiVirusKit 2006
Protection à 100% contre les virus connus et inconnus !
OutbreakShield • Deux modules antivirus • Protection à 100% contre les virus, vers, backdoors, spywares, chevaux de Troie, dialers • Mise à jour horaire
arrivent !
TRUEPREVENT TECHNOLOGIES
Compatible avec
www.dodata.fr
DO DATA

Introduction

Theoretical result (Cohen - 1986)

« Viral detection is an undecidable problem »

- There is no program which would detect every virus.

Introduction

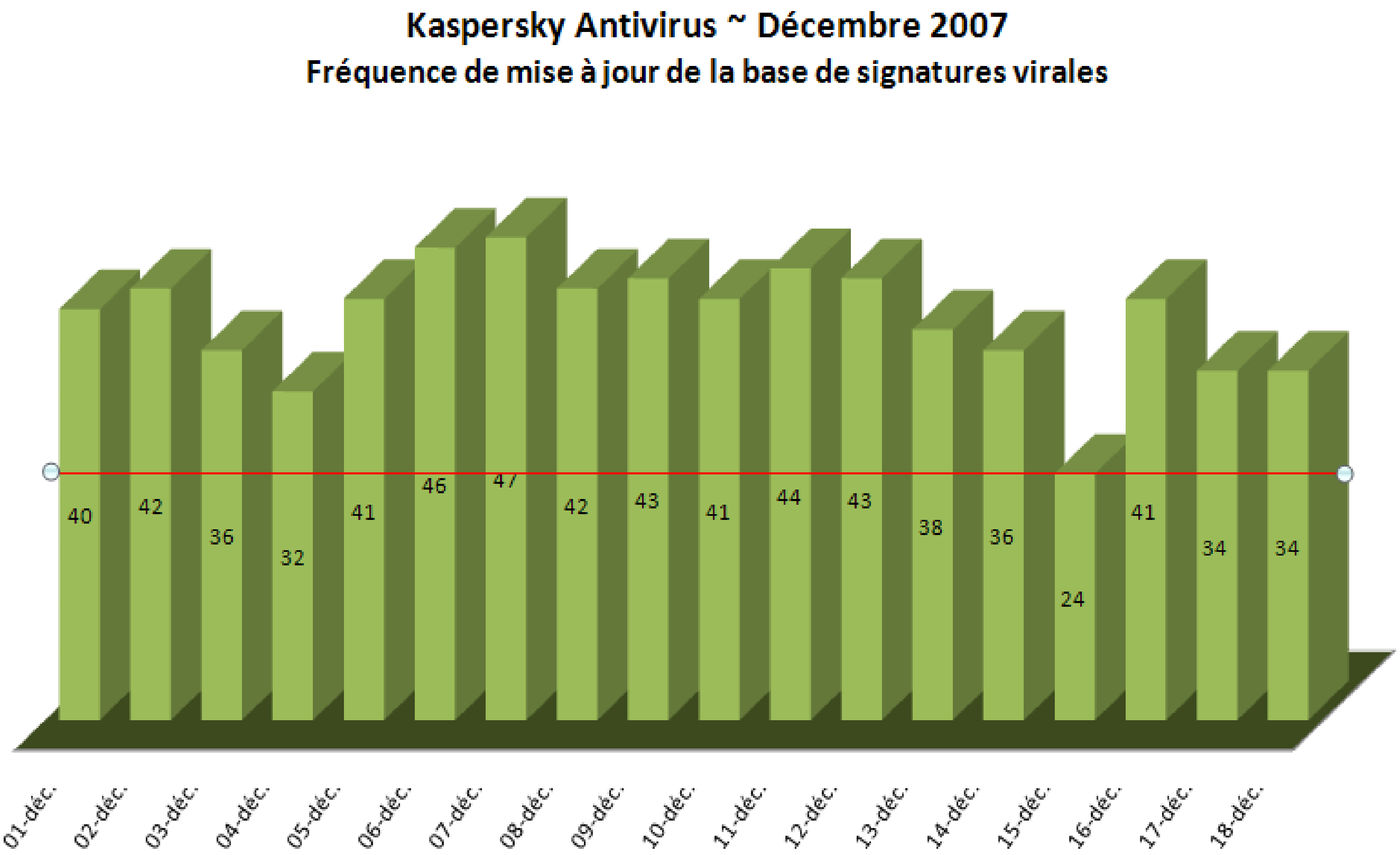
Fact (Attackers' reality)

« Give me a so-called perfect defense or security tool … and I will find how to bypass it somehow ».

- A lot of examples during those recent years (e.g. iPhone security).

Introduction

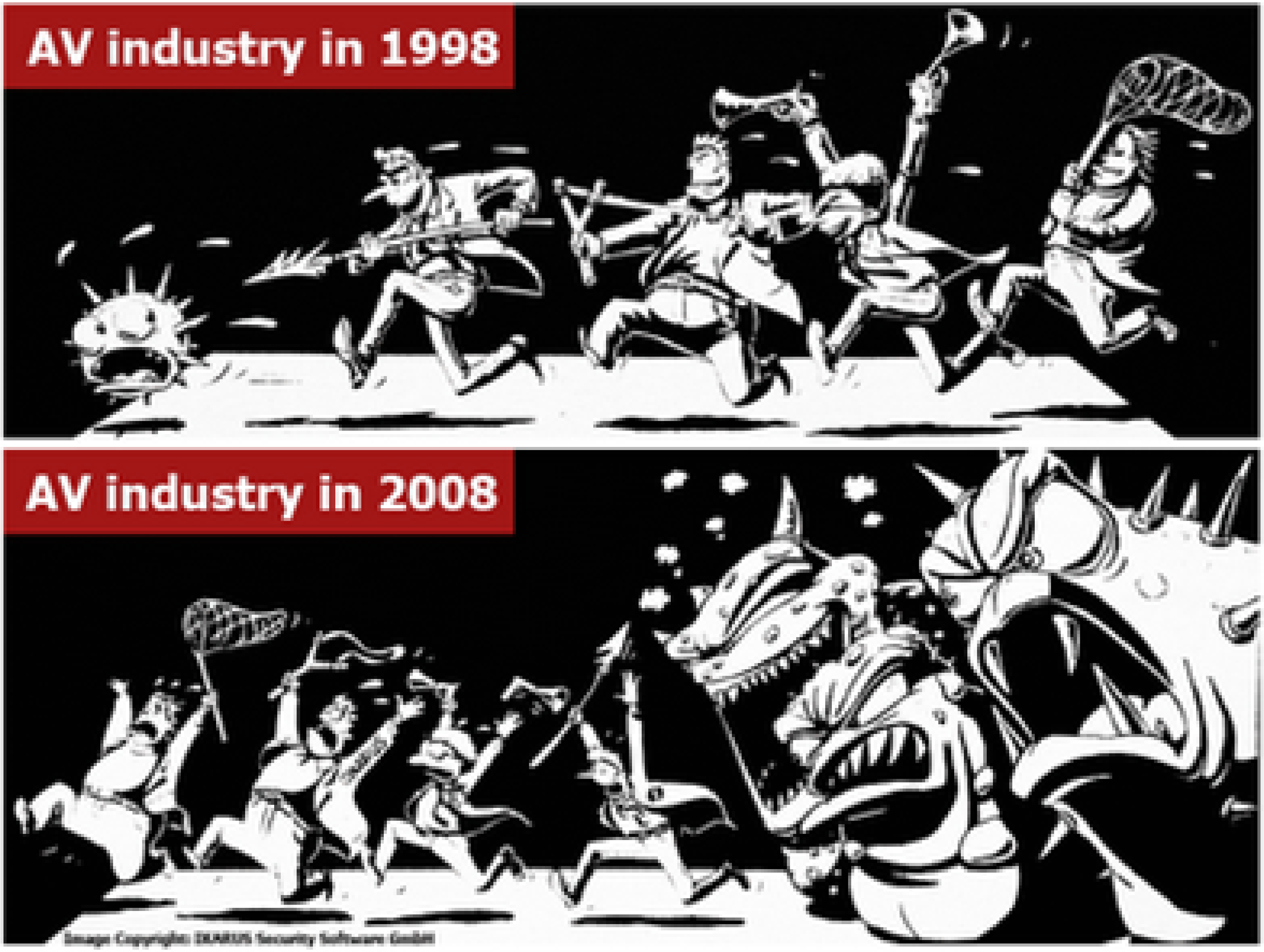
- Who is right? Who is lying? Is there such thing as « winable (computer) war »?
- The answer depends on the kind of attack
  - Wide/Internet-size, popular/generic attacks…
    - ⇒ Best AV software may be right … but the price to pay is high (slow product, high false alarm sensitiveness, frequent updates…).
  - Specific/targeted or small-size attacks
    - ⇒ Attackers are right. AV are totally wrong.
- At the present time, the second case is the most worrying one.



Kaspersky Antivirus ~ Décembre 2007
Fréquence de mise à jour de la base de signatures virales
40
42
36
32
41
46
47
42
43
41
44
43
38
36
24
41
34
34
01-déc.
02-déc.
03-déc.
04-déc.
05-déc.
06-déc.
07-déc.
08-déc.
09-déc.
10-déc.
11-déc.
12-déc.
13-déc.
14-déc.
15-déc.
16-déc.
17-déc.
18-déc.

Introduction

- The real-life situation is worsening.
- Orphan diseases versus large epidemies.
- It is still and it will be always possible to defeat any antivirus technique.
- Basic but critical fact:
  - AV software are commercial product before anything else.
- Let us explain why and how attackers' could design their malware in the future.


AV industry in 1998
AV industry in 2008

Introduction

- This talk is not to promote malware writing!
- Aim of the talk:
  - Understand how the threat is bound to evolve.
  - Be able to understand why AV vendors are wrong.
  - Understand the tools of a « true » computer warfare (or cyberwar).
  - How to prepare prevention and defense.

Summary of the talk

- Introduction.
- Mathematical concepts for dummies (sorry … but it will be not too painful).
- Basic principles of malware design.
- Some examples/cases.
- Conclusion.

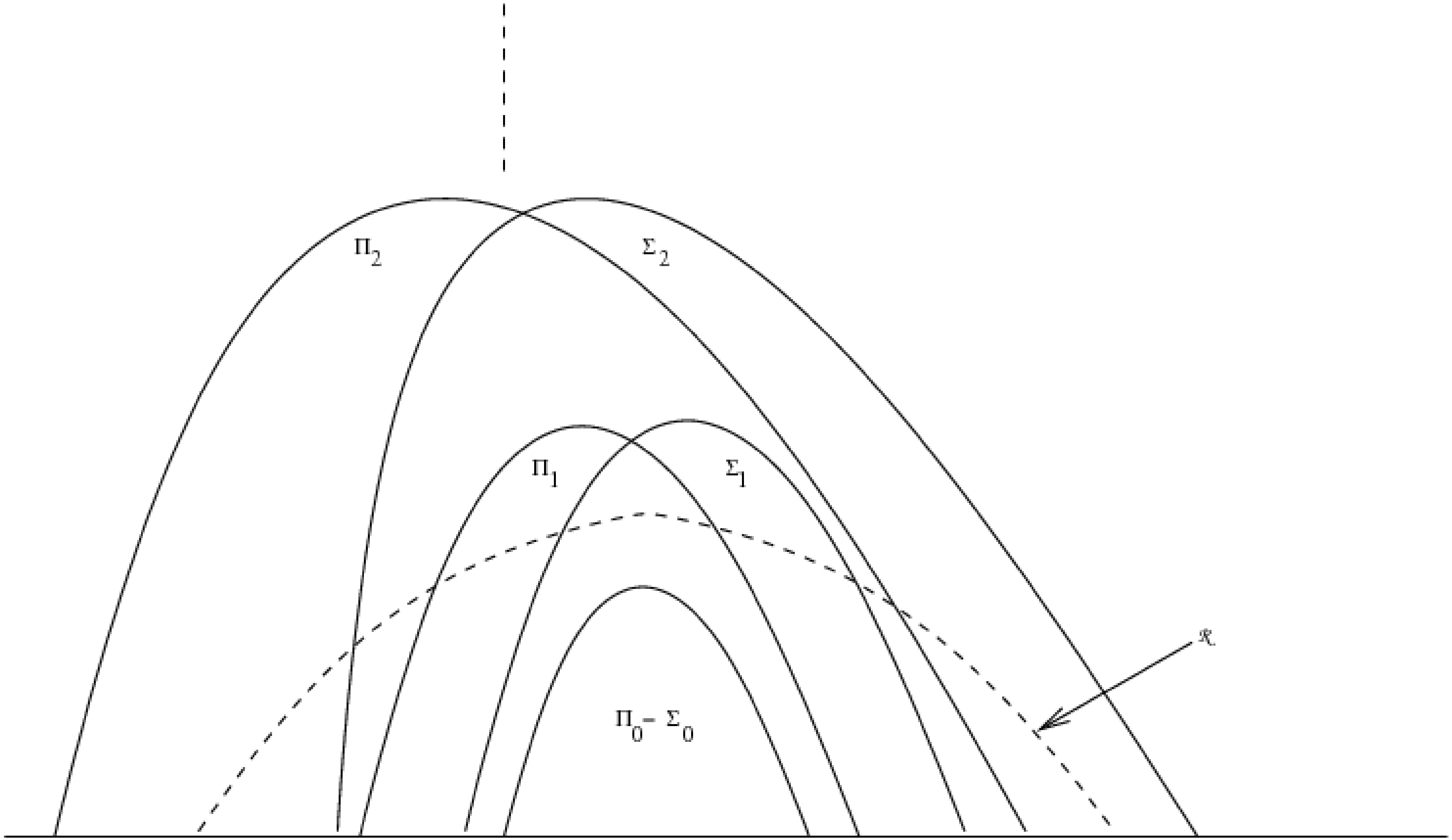
A few mathematical concepts

- **Information theory**
  - Central concept ⇒ entropy.
  - Useful to characterize the amount of information.
  - Any information source can be characterized by its entropy (program, language, data…).
  - For secret quantities, define the amount of secret or of uncertainty.
- **Main tools**
  - Probability theory and statistics.
  - Testing simulability (Filiol - 2007).
    - Tell me which statistical tests you use and my data will behave accordingly to bypass your detection.
  - Cryptology and steganography.

A few mathematical concepts

- **Complexity theory**
  - Central concept ⇒ # of operations to solve a problem.
  - Problems are classified in complexity classes.
    - Polynomial class (P) ⇒ « easy » to solve.
    - Non deterministic polynomial class (NP) ⇒ « hard » to solve.
    - NP-complete ⇒ hardest problems in NP (« very hard »).
    - Even higher complexity classes (Σ_i and Π_i classes with Σ_1 = NP and $\Sigma_2 = NP^{NP}$...).
  - In practice, only the P class is computable (from seconds to a few hours however!).
- Main tools: combinatorics and discrete maths.

A few mathematical concepts



A few mathematical concepts

- **Computability theory**
  - Central concept $\Rightarrow$ Turing machine.
  - Decide whether there exists a Turing machine (e.g. a program) which can compute a given problem.
  - Some problems are not computable (the corresponding Turing machine never stops).
  - Consequently the problem has no solution!
  - Famous example: the virus detection problem!
- **Main tools**: formal grammars and languages.

Basic Principles of (undetectable) Malware Design

Basic Principles of Design

- Build your code in such a way that the problem is (for the AV software):
  - Either « hard » to compute (NP and above),
  - Or is not computable.
- Exploit the fact that AV are commercial products only.
  - AV just devote a few hundreds of cycles only to analyse ⇒ just take more
    - (τ-obfuscation – Beaucamps – Filiol 2006).

Basic Principles of Design (2)

- Fool the detection algorithms.
  - Any detection algorithm can be modelled as a statistical testing (Filiol – Josse 2007).
  - Use testing simulability (Filiol 2007).
  - Use « malicious » cryptography and « malicious » statistics (Filiol – Raynal CanSecWest 2008).
  - Use code armouring to forbid code first analysis
    - Bradley codes (Filiol 2005).
- Imagine new forms of malware.
- And combine all the previous principles!

Basic Principles of Design (3)

- At the code level, think both in terms of:
  - sequence based detection,
  - AND behaviour-based detection.
  - You have to bypass both of them.
  - Example of failure: GpCode (2008).
- Analyze the target (user, AV software, environment…).

A Few Examples and Cases

… among many possible ones

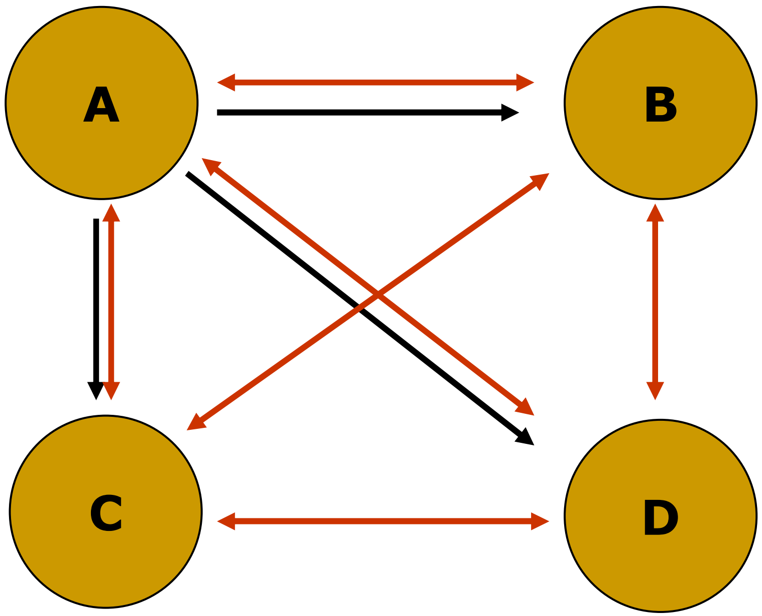
A Few Examples and Cases.

- Let us present a few (among many) examples and cases drawn from
  - Legal cases (forensics analysis).
  - Real targeted attacks analysis.
  - Research and experiments.
- What you MUST keep in mind:
  - Successful attack = Code + attack protocol.
  - Considering the code only can be worthless.
    - In fact think like a military/intelligence guy.

K-ary Malware
or Spliting the Viral Information

K-ary malware.

Starting idea : a real-case (2004)



• The malware installs three variants of itself in memory.
• Variants are light polymorphic versions of A.
• Variants are constantly refreshing themselves (kill, regenerate, mutate and so on…).

Everytime a AV manages to kill one of the variants, the others are reinstalling it.

K-ary malware (formalization - Filiol 2007)

- Definition: family of k (non necessary all executable) files whose union is a malware and whose action is that of a malware. Every part looks innocuitous.
- Two different types:
  - Parallel k-ary malware.
  - Serial k-ary malware.
- Possible to combine the two types:
  - Serial/parallel k-ary malware.

K-ary malware (formalization)

- For every type, three distinct classes:
  - Subclass A (dependent parts).
  - Subclass B (independent parts).
  - Subclass C (weakly dependent parts)
- Validated through different PoC:
  - OpenOffice Virus Final_Touch (de Drézigué at al. 2006).
  - PoC_Serial (Filiol 2007) with $4 \leq k \leq 8$ (any subclass).
  - PoC_Parallel (Filiol 2007) with $k = 4$ (any subclass).
- No detection whatever may be the AV software!

K-ary malware (formalization)

- The detection of k-ary malware has been proven to be at least NP-complete.
  - NP complete if interaction Boolean functions are deterministic.
- It is possible to design still more sophisticated codes:
  - Interaction functions can be non deterministic.
  - Use combinatorial schemes (e.g. threshold schemes).
- Current research work focus on those latter cases.

The Pb_Mot Malware or Generalized Metamorphism.

Basic Principle.

- Is is possible to design a code which cannot be detected ever?
  - The answer is positive provided that you use suitable mutation metamorphic techniques.
  - Consider formal grammars and formal languages.
  - Model your mutation with formal grammar in such a way that detection has to face an undecidable problem.
  - Experimentally validated with respect to sequence-based detection.
  - Current work with respect to behaviour based detection.

Once again mathematics (sorry again).

- Alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$.
- A chain is a sequence of symbols of Σ : $b_1 b_2 b_3 \ldots b_m$ with $b_i \in \Sigma$ and $m \geq 0$.
- If A is a set of chains defined over Σ, we define the set

$$A^* = \{x_1 x_2 \ldots x_n | n \geq 0, x_1, x_2, \ldots, x_n \in A\}.$$

Formal Grammars.

- A formal grammar G is the 4-tuple $G = (N,T, S, R)$ where:
  - N is a set of non-terminal symbols;
  - T is an alphabet of terminal symbols with $N \cap T = \emptyset$;
  - $S \in N$ is the start symbol;
  - R is a rewriting system, that is to say a finite set of rules $R \subseteq (T \cup N)^* \times (T \cup N)^*$, such that $(u, v) \in R \Rightarrow u \notin T^*$ (we cannot rewrite chains which contain only terminal symbols).
- A pair $(u, v) \in R$ is a rewriting rule or production, denoted $u ::= v$ as well.

Rewriting Systems

- A rewriting system R defines a rewriting relation $\Rightarrow_R$ defined as:

  $$rus \Rightarrow rvs \text{ iff } (u, v) \in R \text{ and } (r, s) \in \Sigma^* \times \Sigma^*.$$

- We can build rvs $\in \Sigma^*$ directly from the chain rus $\in \Sigma^*$.
- Example:
  - Take = {A, a, b, c} and R = {(A, aAa), (A, bAb), (A, c), (A, aca)}.
    - A $\Rightarrow_R$ aAa
    - aAa $\Rightarrow_R$ aaAaa
    - aaAaa $\Rightarrow_R$ aacaa

Formal Languages

- A formal language is the set L(G) is the set of "words" generated with respect to the formal grammar G.
- From this point of view, natural languages and programming languages are just instances of a wider concept.
- But there exist far more complex grammars.

Chomsky Classification

- Four main classes of grammars:
  - Class 0 grammars (or *free grammars*). Generate languages decided by Turing machines.
  - Class 1 grammars (or *context-sensitive grammars*). Size of words cannot decrease. This class contains all natural languages.
  - Class 2 grammars (*context-free grammars*). Subsets of this class contain programming languages.
  - Class 3 grammars (or *regular grammars*). Productions are in the form of X ::= x or X ::= xY with $(X,Y) \in N^2$ and $x \in T^*$.
- There exist other (still more complex) formal grammars.

Formal Definition of Code Mutation

- Consider the set of x86 instructions as the working alphabet.
- Instructions may be combined according to (rewriting) rules that completely define every compiler.
- This set of rules can be defined as a class 2 formal grammar (assembly language).
- Implementing a polymorphic engine consists in generating a formal language: the polymorphic language with its own grammar.
  - $\Rightarrow$ E.g. Polymorphic grammar.

Trivial Polymorphism.

- Take the grammar G = {{A,B}, {a, b, c, d, x, y}, S, R}.
- Instructions a, b, c and d are garbage code while instructions x and y are the decryptor's instructions. R is defined as:
  - S ::= aS|bS|cS|xA
  - A ::= aA|bA|cA|dA|yB
  - B ::= aB|bB|cB|dB|ε
- This polymorphic language is made up of every word in the form of

$$\{a, b, c, d\}^*x\{a, b, c, d\}^*y\{a, b, c, d\}^*$$

Formal Definition of Code Mutation (2)

- Every of the language words corresponds to a mutated variant of the initial decryptor.
- It is “easy” (e.g for an antivirus) to determine that the word *abcddxd* is not in this language with respect to G, contrary to the word *adcbxaddbydab*.
- The critical issue for any antivirus is then to have an algorithm which is able to determine whether a “word” (a mutated form) belongs to a polymorphic language or not.
- What is the detection complexity (or language decision)?

Langage Decision Problem

- Definition: Let G = (N,T, S, R) be a grammar and $x \in T^*$ a chain with respect to G. The language decision problem with respect to G consists in determining whether $x \in L(G)$ or not.
- To solve the language decision problem, we can consider
  - Deterministic Finite Automata (DFA),
  - Non deterministic Finite Automaton (NFA),
  - Turing machines.

Langage Decision Problem vs Detection

- If an antivirus embeds an automaton A that can solve the (polymorphic) language decision problem with respect to a given polymorphic grammar, then detection is possible.
- Two critical issues are then to be considered:
  - the relevant complexity of the automaton,
  - every time the polymorphic grammar is changing, the antivirus software must be upgraded with a new automaton which decides the new polymorphic language.
- Metamorphic techniques are more powerful than polymorphic ones since every new metamorphic mutation produces a new grammar and a new word generated by the latter at the same time.

Formal Definition of Metamorphism

- Definition: Let $G_1 = (N, T, S, R)$ and $G_2 = (N', T', S', R')$ be grammars where T' is a set of formal grammars, S' is the (starting) grammar G_1 and R' a rewriting system with respect $(N' \cup T')^*$. A metamorphic virus is thus described by G_2 and every of its mutated form is a word in $L(L(G_2))$.
- This definition describes the fact that from one metamorphic form to another, the virus kernel is changing: the virus mutates and changes the mutation rules at the same time.
- Detecting such sophisticated metamorphism is equivalent to solve the language decision problem twice.

Language Decision Complexity

- Theorem: The language decision problem:
  - is undecidable for class 0 grammars;
  - has NP-complexity for class 1 and class2 grammars;
  - has polynomial complexity for class 3 grammars.
- Then the choice of underlying grammar is essential when designing a polymorphic/metamorphic engine. It has a direct impact on its resistance against its potential detection.

The PoC Pb_Mot Metamorphic Malware.

- Proof-of-concept of undetectable metamorphic malware.
- Based on the « Word problem » defined by Post in 1950.
  - One of the most famous undecidable problems.
  - Are two finite words r and s over Σ equivalent or not, up to a rewriting system R.
- Equivalently, it consists in deciding whether $r \Rightarrow_R^* s$ or not.

Tzeitzin Systems.

- Smallest undecidable semi-Thue systems T_0 and T_1:

(ac, ca),
(ad, da),
(bc, cb),
(bd, db),
(eca, ce),
(edb, de),
(cca, ccae)

(ac, ca),
(ad, da),
(bc, cb),
(bd, db),
(eca, ce),
(edb, de),
(cdca, cdcae),
(caaa, aaa),
(daaa, aaa)

The PoC Pb_Mot Metamorphic Malware (2).

- Use formal grammars whose rewriting system contains a Tzeitsin systems.
  - $\Rightarrow$ the code mutation engine will be undecidable as well.
- The engine's rewriting (mutation) rules change from mutation to mutation.
- Two main constraints are to be satisfied:
  - the rewriting system of G_2 contains an undecidable Thue system;
  - every word (hence a grammar) in L_i (G_2), during the i^{th} mutation step, contains an undecidable Thue system as well.
- The rewriting system of L_i (G_2) grammars are coded as words on the alphabet $(N \cup T)^*$.
- Detection of PoC Pb_Mot is undecidable

Discussion

- What about the detection of PoC Pb_mot metamorphic codes?
  - Sequence-based detection fail since mutation is based on an undecidable problem.
  - On execution, once the code is unprotected, it can be analysed. But antivirus and virus do not to play the same game.
  - With τ -obfuscation (Beaucamps - Filiol, 2006), metamorphic codes can delay their own disassembly in an arbitrary time τ , more than any antivirus (commercial products) can accept.

Discussion (2)

- The theoretical approach with formal grammars is a new, promising way to systematically distinguish efficient techniques from non trivial or unefficient ones.
- Until now, known (theoretically detected) metamorphic codes refer to rather naive or trivial instances for which detection remains “easy”.
- Some behaviours may represent useful invariant that could be considered by antivirus in the future (behaviour-based detection).
- Nest step is behavioural polymorphism/metamorphism: code behaviours both at the micro- and the macro level would change from replication to replication.
- Systematic exploration of subclasses of grammar is essential as well.

Optimized worm propagation.

…or how to design the perfect botnet.

Optimized worm propagation.

- How to design a stealth but fast enough worm to subvert an unknown Internet-sized network?
  - Design of a two-level malicious network.
  - Use some combinatorial structure to spread and manage the worm.
  - The worm does not require any *a priori* knowledge about the network.
- The level of connection overhead (wrong, useless worm connections) is optimally lowered.
- PoC and SuWast (simulator) (Filiol and al. 2007)

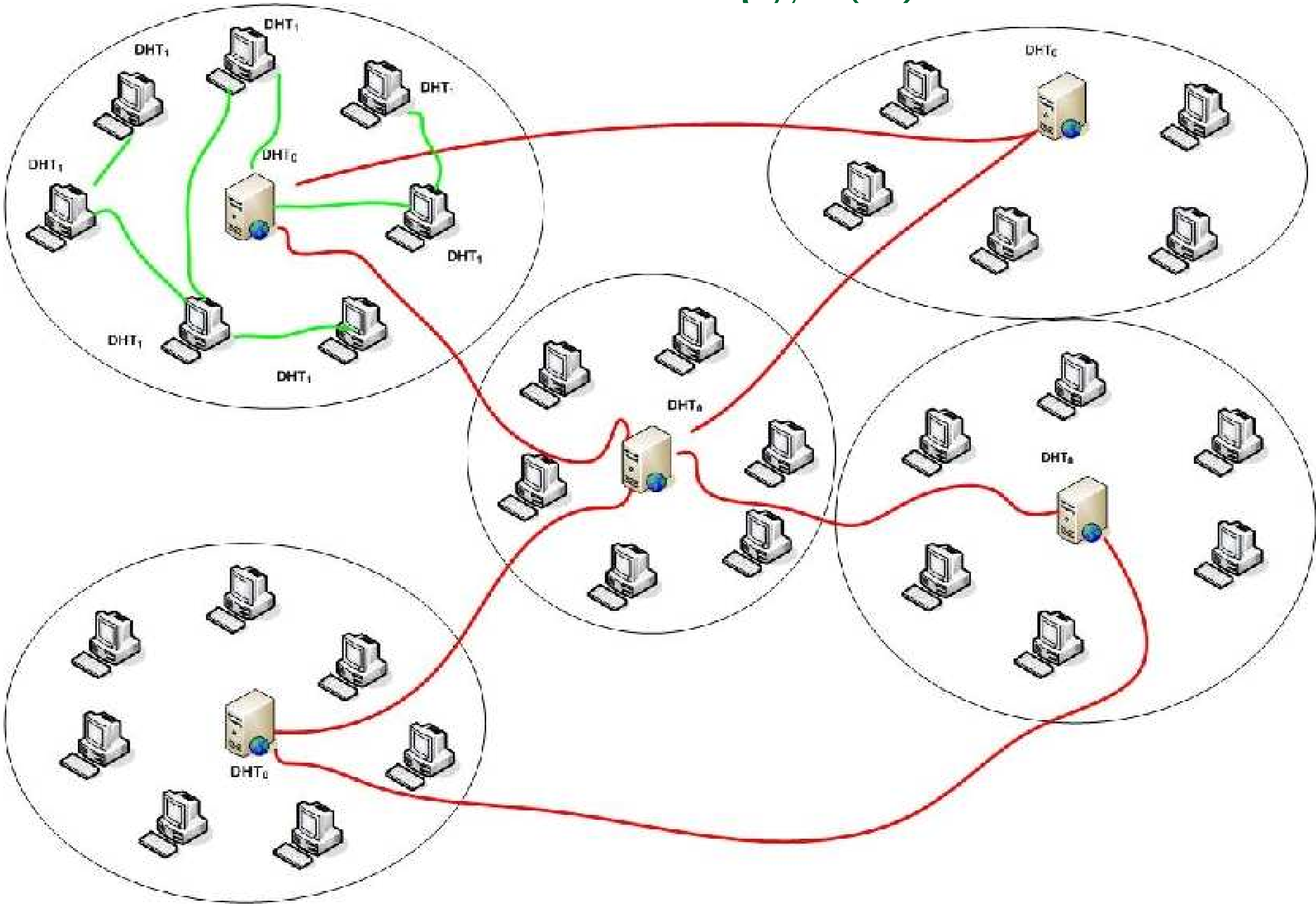
General Worm Strategy.

- The target network is set up into a two-level hierarchy.
  - Locally, « malicious » P2P networks are set up (lower networks; local maganement of dynamic address hosts).
  - Every malicious lower network also manage a single static IP adress.
  - At a macro level, a malicious network of static IP addresses is set up (worm upper network).
  - Globally, a graph structure G to manage fixed IP addresses only (maintained at the attacker's side).
- The basic tools to manage the different networks are DHT (Dynamic Hash Tables).

General Worm Strategy (2).

- These two structures are connected at the fixed IP addresses' level.
- The attacker monitors data sent by every infected machine.
- The overall, upper level topology of the malicious network is managed at the attacker's level through the graph G.
- The two-level structure aims at making the worm spread as invisible as possible.
  - From one given node, the worm spreads to nodes that used to communicate with it only.
  - Existing previous connection is considered as a "trust" relation.

General Worm Strategy (2).



•
ζ
s

Worm Spread Mechanism.

This step aims at finding IP addresses to infect.

1. With a probability $p_0 < 0.1$, generate a random IP adress. Then, the worm tries to infect this random IP address.
2. The worm then locally looks for existing addresses to infect:
   - ARP table and directory of given software applications: Internet browser, antivirus, firewall...
   - Identification of machines already connected to the local machine: *netstat*, *nbtstat*, *nslookup*, *tracert* ...
3. Attempt to spread to these addresses and update DHT structures if successful.
4. Information is sent to the attacker’s monitoring machine.

The worm determines whether a target is already infected or not.

Collected Data.

- To monitor the worm activity and to evaluate its efficiency, the attacker use some indicators.
- The corresponding (directed) graph structure G (describes the worm upper network) is defined as follows:
  - each fixed IP address is a graph node,
  - node i is connected to node j if machine j has been infected by machine i .

Collected Data (2).

- Let us suppose that machine i successfully managed to infect machine j at time t. The following data are collected:
  - IP address of machine i .
  - IP address of machine j .
  - A single fixed IP address.
  - The time of infection.
  - The infection mark (machine j was already infected or not)

Managing the Infected Network

- Once the worm has infected any possible machine, the attacker has to control, set up or modify the worm behavior (botnet admin).
  - DHT structures must be managed in order to avoid a too much increase of their size.
  - Systematically, the DHTs of a given machine i dynamically manages and keeps only the IP addresses corresponding to machines recently connected to machine i .
- Use of a node identification system based on node ID built from the local IP address and the XOR metrics.

Managing the Infected Network (2)

- Use of a weighted measure for every IP address in the DHTs tables. Let us consider DHT^i_1 of machine i .
  - For every other IP address j in DHT^i_1 , let us denote d_{ij} the (xor) distance between machines i and j and t_{ij} the last connection time (in seconds) between machine i and j .
  - Consider the following weight:

$$w_{ij} = d_{ij} \times t_{ij} .$$

- So, DHT^i_1 permanently self-updates in order to keep only the IP addresses with lowest weight w_{ij} .

The Botnet Graph

- The aim is to model the connections between fixed addresses by means of a directed graph G.
  - nodes of G, denoted $(n_i)_{1 \leq i \leq N}$ are representing fixed IP addresses (generally a server) ;
- Entries of the incidence matrix of G are defined by:
  - $a_{i,j} = 1$ if computer j has been infected by computer i
  - Otherwise $a_{i,j} = 0$.

Managing the Infected Network (3)

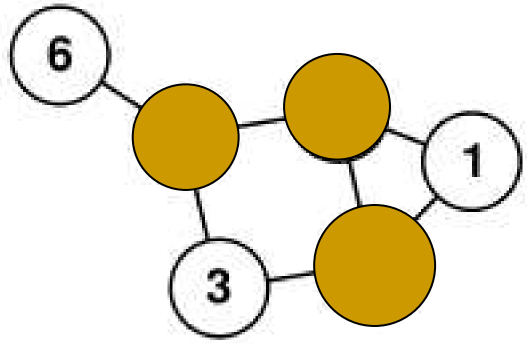
- Search for vertex cover within the graph.
- Definition: Let G a undirected graph (V , E). The vertex cover is a subset V ′ of the vertices of the graph which contains at least one of the two endpoints of each edge:

$$V' \subset V : \forall \{a, b\} \in E, a \in V' \text{ or } b \in V'$$

- The vertex cover problem is NP-complete.
- But efficient heuristics do exist (Dharwadker 2006).

Managing the Infected Network (4)

- Let us consider the following toy graph.



- The node subset {2, 4, 5} is a vertex cover of G. Moreover, it is the smallest possible one.

Managing the Infected Network (4)

- From the data collected the attacker will first try to identify a vertex cover.
  1. The attacker looks for a vertex cover $V' = \{n_{i1}, \ldots, n_{ik}\}$. He may consider a partial subgraph.
  2. The information that intends to adapt the worm behaviour is sent to nodes $n_{ij} \in V'$ with $1 \leq k$, only.
  3. Each of the nodes $n_{ij} \in V'$ will then spread locally to other nodes of the graph according to a suitable ordering (for exemple, in the previous node 3 can be updated either by node 2 or node 4, but only node 2 will).
- The use of a vertex cover set minimizes the number of communications between nodes while covering all the nodes quite simultaneously.
- From the network defender's side, the problem is far more complex since he does not have the collected data in the same way the botherder does.

Simulation and results

- Design of Suwast (*Super Worm Analysis and Simulation Tool*).
- Non public simulator.
- Powerful simulation tool of complex, heterogenous networks (clients, servers, routers...), enabling simulations of network attacks in a controlled environment at packet level.
- Large-scale simulations (up to a 60,000-host heterogeneous network on a single 2 GB machine).
- Possibility to interconnect such machines to simulate heterogeneous networks of millions of hosts.

Simulation and results (2)

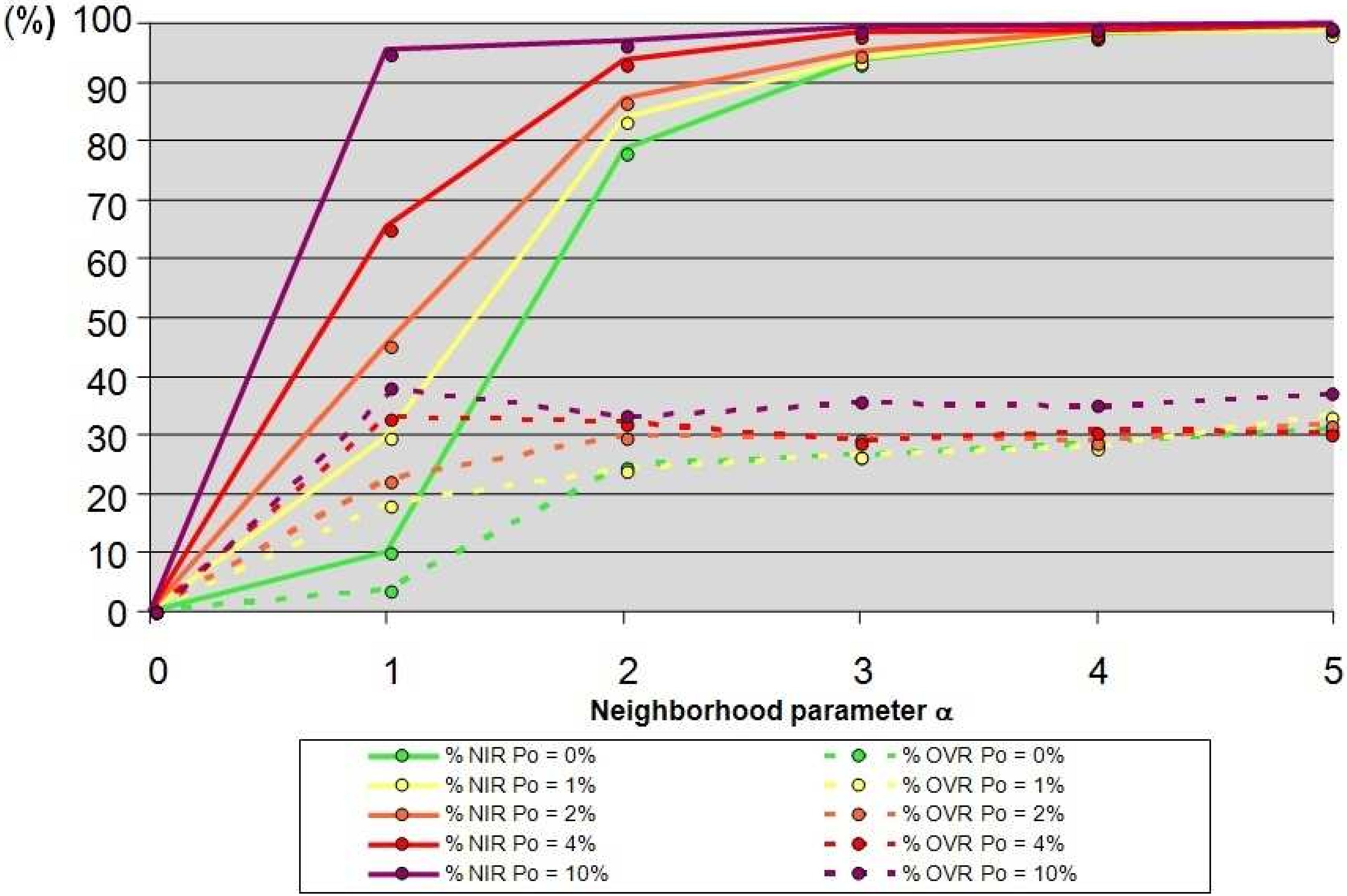
- Two metrics have been used:
  - the Network Infection Rate (NIR):

$$NIR = \frac{\#\ \text{of infected hosts}}{N}$$

  - the Overinfection Rate (OR):

$$OR = \frac{\#\ \text{of infection attempts of already infected hosts}}{\#\ \text{of infected hosts}}$$

NIR and OVR (%) on a 100-server network



Simulation and results (3)

- Three essential results are noticeable:
  - the parameter p_0 has a significant impact on both the NIR and the OR. The case $p_0 = 0.04$ is optimal, provided that the server neighborhood parameter is not to large;
  - the NIR is systematically greater to 90 % if $3 \leq \alpha$ (server neighborhood parameter), most of the results being closer to 99 %.
  - the server neighborhood parameter α has a more significant impact on the OR. Optimally, we have

$$\alpha \in [3, 6].$$

Conclusion

- Quite an infinite number of doing undetectable malware.
- What is the level of threat nowadays?
  - Quite impossible to say.
  - Potentially high for targeted attacks (intelligence agencies or military forces in some countries).
  - Probably low to medium for other attackers… until now.
  - Require skilled malware writers with a good level both in mathematics, computer science and programming.

Conclusion

- The solution to fight against those malware of the future is no longer technical and will never be!
- Only accurate and strong security policies are likely to be the best protection.
  - Avoid to be infected or you are dead!

Thanks for your attention

Have a nice Hack.lu conference

Bibliography

- E. Filiol (2007) *Techniques virales avancées*, collection IRIS, Springer Verlag. An English translation is pending.
- P. Beaucamps et E. Filiol. On the possibility of practically obfuscating programs - Towards a unified perspective of code protection. WTCV'06 Special Issue, G. Bonfante & J.-Y. Marion eds, *Journal in Computer Virology*, 3 (1), 2007.
- E. Filiol, S. Josse. A Statistical Model for Viral Detection Undecidability. EICAR 2007 Special Issue, V. Broucek ed., *Journal in Computer Virology*, 3 (2).
- E. Filiol. Formalization and Implementation Aspects of K-ary (malicious) Codes. EICAR 2007 Special Issue, V. Broucek ed., Journal in Computer Virology, 3 (2).
- E. Filiol. Metamorphism, Formal Grammars and Undecidable Code Mutation. *International Journal in Computer Science*, 2 (1), pp. 70--75, 2007.
- E. Filiol, E. Franc, A. Gubbioli, B. Moquet et G. Roblot. Combinatorial Optimisation of Worm Propagation on an Unknown Network. *International Journal in Computer Science*, 2 (2), pp. 124--130, 2007.
- E. Filiol & F. Raynal. Malicioux Cryptography ... reloaded and also malicious statistics. CanSecWest 2008, Vancouver, March 26th - 28th, 2008.