Noisy Defenses: Subverting Malware's OODA loop

Daniel Bilar

Wellesley College
Department of Computer Science

May 9, 2008
CSIIRW '08, Oak Ridge National Labs

Talk roadmap

Status Quo

Classic AV byte-pattern matching may have reached its practical and theoretical limits with present modern malware

Research 2005-2007: Structural fingerprinting

Can statistical structural 'fingerprinting' assist MW identification, classification? Approaches included Win32 API calls, opcode frequency distribution and callgraph properties

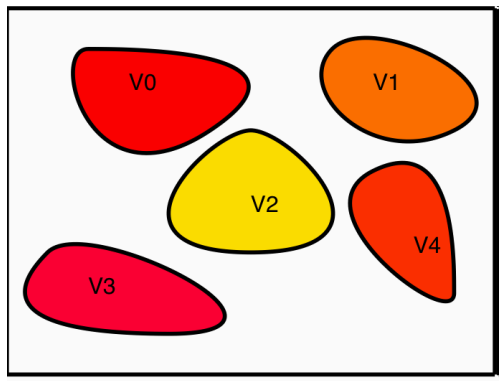
Moving forward: Noisy Defenses

Enter malware's decision cycle and adapt responses to influence malware's next steps.
Comprehensively control entropy of targets (passive defense) and environments (active defense) to manipulate malware's OODA loop

Malware taxonomy: Poly- and Metamorphic Malware

Description

Polymorphic malware contain decryption routines which decrypt encrypted constant parts of body.
Metamorphic malware generally do not use encryption, but mutates forms in subsequent generations.



Metamorphic virus
semantically equivalent

Figure: Mutated body in subsequent generations

Malware taxonomy: Interactive Malware

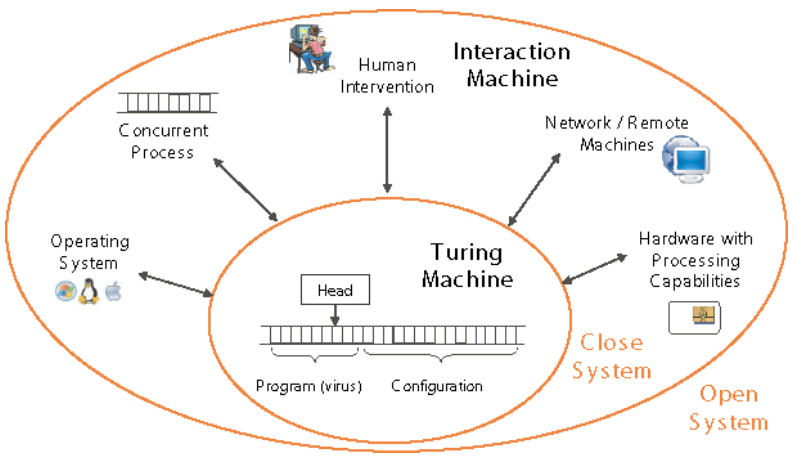


Figure: Interactive Malware: Open vs closed system. Interacts with OS, network, other programs, humans. May also use dissimulation techniques. Picture by Grégoire Jacob, French Army Signal Academy ESAT (France)

Metamorphic/Polymorphic Detection

Empirical AV Results

Date	#MW Family	#AV Scanners	Miss (%)
2008/02	8	15	27
2007/08	10	15	25
2007/02	10	15	38
2006/08	10	15	48

Table: Updated AV scanners against hundreds of well-known, previously submitted, highly polymorphic and metamorphic malware samples (AV-Comparatives.org)

Research 2005-2007: Statistical Structural Fingerprinting

Structural Perspective	Description	Statistical Fingerprint
Assembly instructions	Tabulate instructions	Opcode frequency distribution
Win32 API calls	Observe API calls made	API call vector
Callgraph	Explore control flow of functions	Graph structural properties

1st Fingerprint: Win32 API Calls

Synopsis

Observe and record Win32 API calls made by malicious code during execution, then compare them to calls made by other malicious code to find similarities

Goal

Classify malware quickly into a family
Set of variants make up a family

Main Result (2005) [Rie05]

Simple (tuned) Vector Space Model yields over 80% correct classification

Win32 API calls: Results

Family	# of members	Discrepancies # correct
Apost	1	0
Banker	4	4
Nibu	1	1
Tarno	2	2
Beagle	15	14
Blaster	1	1
Frethem	3	2
Gibe	1	1
Inor	2	0
Klez	1	1
Mitgleider	2	2
MyDoom	10	8
MyLife	5	5
Netsky	8	8
Sasser	3	2
SDBot	8	5
Moega	3	3
Randex	2	1
Spybot	1	0
Pestlogger	1	1
Welchia	6	6

Figure: 77 malware samples classified

Misclassifications

Threshold	☑ %	☑ #	false fam.	both	miss. fam.
0.7	0.8	62	5	8	2
0.75	0.8	62	5	6	4
0.8	0.82	63	3	6	5
0.85	0.82	63	2	4	8
0.9	0.79	61	1	4	10
0.95	0.79	61	2	3	11
0.99	0.62	48	0	2	27

Figure: Threshold parameter

Classification and AV corroboration

Classification by 17 AV scanners yields 21 families. > 80 % correspondence (*csm* threshold $= 0.8$).

2nd Fingerprint: Opcode Frequency

Synopsis

Statically disassemble the binary, tabulate the opcode frequencies and construct a statistical fingerprint with a subset of said opcodes

Goal

Compare opcode fingerprint across non-malicious software and malware classes for quick identification purposes

Main Result (2006) [Bil07b]

For differentiation purposes, infrequent opcodes explain more data variation than common ones

Opcode frequency: Results

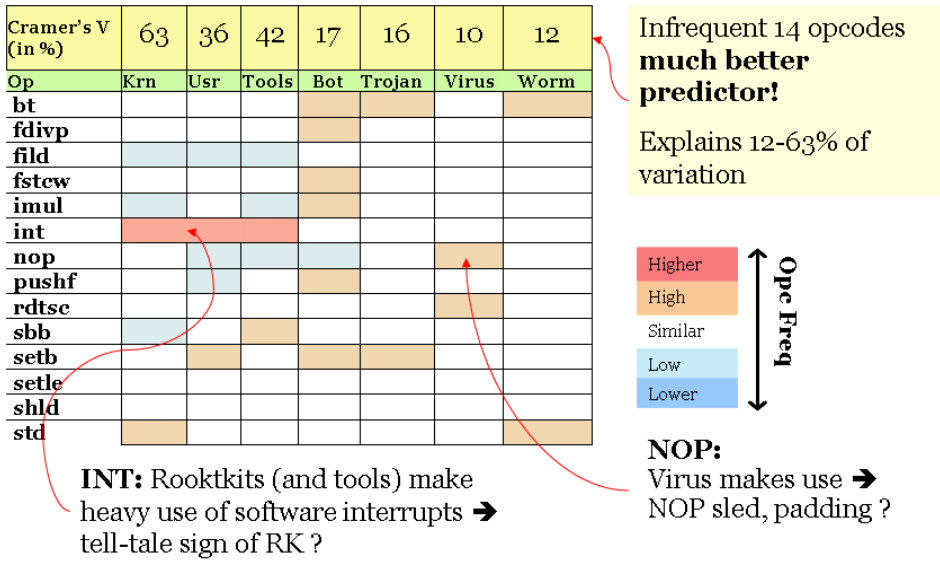


Figure: Infrequent opcodes explain more variation, hence better predictor for malware differentiation

3rd Fingerprint: Graph properties

Synopsis

Represent executables as callgraph, and construct graph-structural fingerprint for software classes

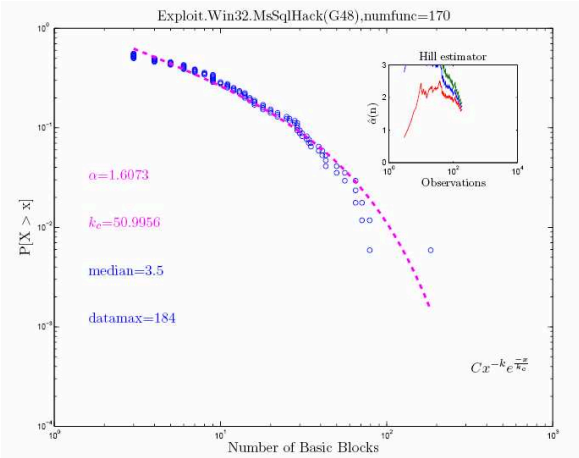
Goal

Compare 'graph structure' fingerprint of unknown binaries across non-malicious software and malware classes

Main Result (2007) [Bil07a]

Malware tends to have a lower basic block count, implying a simpler structure: Less interaction, fewer branches, limited functionality

Callgraph: Degree Distribution



(b) MW sample: Fitting α_{bb} and k_c

Figure: Pareto fitted ECCDF with Hill estimator $\hat{\alpha}(n)$

Power (Pareto) law

Investigate whether indegree $d_{indeg}(f)$, outdegree $d_{outdeg}(f)$ and basic back count $d_{bb}(f)$ distributions of executable's functions follows a truncated power law of form

$$P_{d_*(f)}(m) \sim m^{\alpha_{d_*(f)}} e^{-\frac{m}{k_c}}$$

with α a power law exponent, k_c distribution cutoff point, $\hat{\alpha}(n)$ Hill estimator (inset) used for consistency check

Callgraph: Differentiation Results

class	Basic Block	Indegree	Outdegree
t	2.57	1.04	-0.47
Goodware	N(1.634,0.3)	N(2.02, 0.3)	N(1.69,0.307)
Malware	N(1.7,0.3)	N(2.08,0.45)	N (1.68,0.35)

Table: Only one statistically relevant difference found: Basic block distribution metric $\mu_{malware}(k_{bb}) \neq \mu_{goodware}(k_{bb})$ via Wilcoxon Rank Sum

Interpretation

Malware tends to have **a lower basic block count**, implying a simpler structure: Less interaction, fewer branches, limited functionality

Quo vadis, AV?

Modern Malware Detection Complexity

Bad Classic 'white-box' AV has been failing for maybe half a decade. Structural fingerprinting is of no direct help (alas).

Worse Practical AV detection in hitherto tractable *linear time* struggling against *NP-completeness* and *undecidability*. Detection of interactive malware is at least in class $\sum_3$, no longer $\prod_2$ [JF08]

Conjecture

Detection of modern malware through techniques based on (strong) Church-Turing thesis (computation-as-functions) a practical/theoretical dead-end

Moving forward

Posit need for shift towards 'interactive computation', both in practice and theory

Interactive Computation [GW05]

Idea

Theoretical bridge between TM (functions) and concurrency (communication) aspects

Back to the Future: Elucidated by Turing

Turing proposed interactive choice machines as another model of computation distinct from automatic TMs - and not reducible to them.

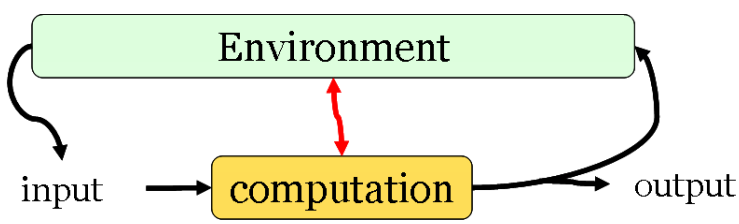


Figure: Open I/O *during* computation, not just before or after

Information-Gain Adversarial Malware

Entropy

Byte sequence-matching AV approach
Input (suspicious data) → function (pattern-based decision) → output (yes/no malicious)

Modern malware systematically thwarts AV information gain through multi-stage, interactive, opaque implementation

How to defend?

Two can play the information thwarting game Adapt environment /defenses to control malware's information gain for benefit of defender

Noisy Defense Idea

Enter malware's decision cycle and adapt responses to influence malware's next steps. Comprehensively control entropy of targets (passive defense) and environments (active defense) to enter malware's OODA loop.

Passive and Active Defenses

Passive Defenses

Approach Prevent exact identification of targets and environment by introducing irregularities

Components

Network/Host Visualization, honeynets (simulated decoys that detract from 'real' ones)
Program/OS Hot-patch binaries, ASLR (random heap, stack, library positioning)
Implementation Mitigate side channel attacks (data structure, protocols, observables)

Active Defenses

Approach Model malware's internal hypothesis structure, then control decisions by manipulating view of the projected world

Components

Observation Framework
Infer MW's internal hypothesis structure via dynamic 'black-box' interaction

Control Framework
Dynamically choose strategies that control adversarial information gain for benefit of defense

Active Defense Framework

Observe and Choose Model: PQS

Process Query System DBMS framework that allows for process description queries against internal models (FSM, Hidden Markov, etc) Designed to solve the Discrete Source Separation Problem [CB04]

Control Information Gain: Play Games

Create illusion of win-win situation viz the malware's goals by weakening useful/accurate and strengthening useless/misleading information gain

Idea

Detect processes by leveraging correlation between observations and processes' states
Processes detection Models of MW's internal control structure
State estimation Estimate the current control flow 'state' the malware is in

Idea

Play repeated, Bayesian, imperfect-information games to probabilistically identify entities, then infer/estimate its control structure, then use OODA-loop controlling strategies

Active Defense: Illustration with Toy Example

Suppose malware has been (perhaps probabilistically) identified through interaction and observation. Its toy internal hypothesis structure (and strategies) are modeled by Scan;if XP penetrate;if filtered DoS in a PQS internal model. The defense's strategies are S (no defense), S_{HP} (honeypot), S_{FLT} (filter/block ICMP response). The game matrix shows (hypothetical) payoffs of the defense's and malware's strategy combinations.
The malware starts scanning and wants to get to [Pen, S] penetrating a real host. The defense wants to engage sequential strategies such that the malware penetrates a fake host [Pen, S_{HP}], thereby giving the illusion of a win for the malware while learning more about it. Again, the defense wants to iteratively control, not necessarily minimize malware's hoped-for entropy reduction. Strategies may not be fixed and dynamically generated as PQS models adapt to the malware responses, as denoted by $S_{\ldots}$ (new defense strategy) and $< UO >$ (unknown observation).

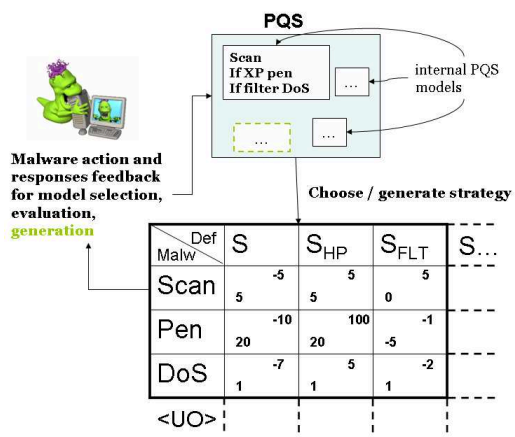


Figure: Game loop, PQS modeling example malware's control structure Scan;if XP penetrate;if filtered DoS

Thoughts

Open Questions

PQS models
Dynamic inference to model malware's hypothesis structure is open area.
Formalization of interactive malware started just very recently.

Suitable System State Description and Entropy Measures
Defense goal is not to maximally confuse malware, but to manipulate malware's decision tree by controlling its cross-entropy calculus D^x of perceived target/enviroment.

Requires *appropriate state representation of targets/enviroments*, since this directly determines cross-entropy measure D^x.

Ex: If system's governing distribution (probability of given realization)
$\mathbb{P} = P(n_i|q_i, N, s, I)$ s.t. prior probabilities q_i, number of entities N, number of states s with $\sum_{i=1}^{s} n_i = N$ and background information I is *multinomial* with $\mathbb{P} = N! \prod_{i=1}^{s} \frac{q_i^{n_i}}{n_i!}$,
then cross-entropy to manipulate is Kullback-Leibler
$D^x_{KL} = \sum_{i=1}^{s} \left(p_i N^{-1} \ln N! + p_i \ln q_i - N^{-1} \ln((p_i N)!)\right)$.
However, if system is not governed by multinomial $\mathbb{P}$ (e.g Bose-Einstein system's $\mathbb{P}_{BE}$ is multivariate negative hypergeometric), D^x_{BE} is KL only in special cases [Niv07].

Default Distrust?
When to engage active entropic defense? Arguably from t_0 - then all interactions (genuinely benign and malicious) are assumed a priori potentially malicious.
Interactions as validation mechanism [CJB99]. Feasible, workable?

References

- Daniel Bilar, *On callgraphs and generative mechanisms*, J. Comp. Vir. **3** (2007), no. 4.
- ———, *Opcodes as predictor for malware*, Int. J. Electron. Secur. Digit. Forensic **1** (2007), no. 2.
- George Cybenko and Vincent Berk, *An overview of process query systems*, Proc. SPIE, vol. 5403, 2004.
- George Cybenko, Guofei Jiang, and Daniel Bilar, *Machine Learning Applications in Grid Computing*, Proc Ann Allerton Conf CCC **37** (1999).
- Dina Goldin and Peter Wegner, *The Church-Turing Thesis: Breaking the Myth*, LNCS: New Computational Paradigms (2005), 152–168.
- Gregoire Jacob and Eric Filiol, *Malware As Interaction Machines*, J. Comp. Vir. **4** (2008), no. 2.
- Robert K. Niven, *Combinatorial Information Theory: I. Philosophical Basis of Cross-Entropy and Entropy*, ArXiv (2007).
- Chris Ries, *Automated identification of malicious code variants*, J. Comput. Small Coll. **20** (2005), no. 5, 140–141.