

Flying below the Radar: What modern malware tells us

Daniel Bilar
Department of Computer Science
Wellesley College
Wellesley (MA), USA

dbilar <at> wellesley dot edu

Talking about engineering and theory challenges for malware to come
Prediction is very hard, especially about the future

Overview

Classic AV pattern matching identification may have reached its practical and theoretical limits with present modern MW

Results of author's structural MW analysis
Propose moving towards iterative games and black-box process modeling, as expressed by interactive computing models

MW to come? k-ary, Satan, IC and Quantum

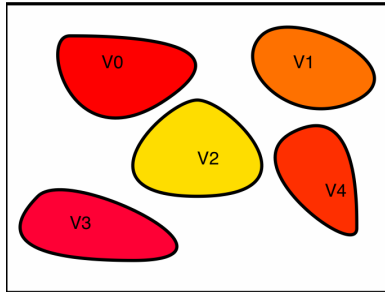
Metamorphism/Polymorphism

Polymorphic malware contain decryption routines which decrypt encrypted constant parts of body

Metamorphic malware generally do not use encryption, but mutate body in subsequent generations

Confusing: Sometimes, polymorphism subsumes both terms, especially in 'older' work

Metamorphic Detection



Metamorphic virus
semantically equivalent

In Feb 2007, 17 state-of-the-art AV scanners checked against 12 well-known, previously submitted, highly polymorphic and metamorphic malware samples.

Detection miss rates:
100% to 0%, ave of 38%.

Metamorphism mutate body
in subsequent generations

Today: Highly variable MW

Bagle/Beagle: email-born worm, first appeared in 2004

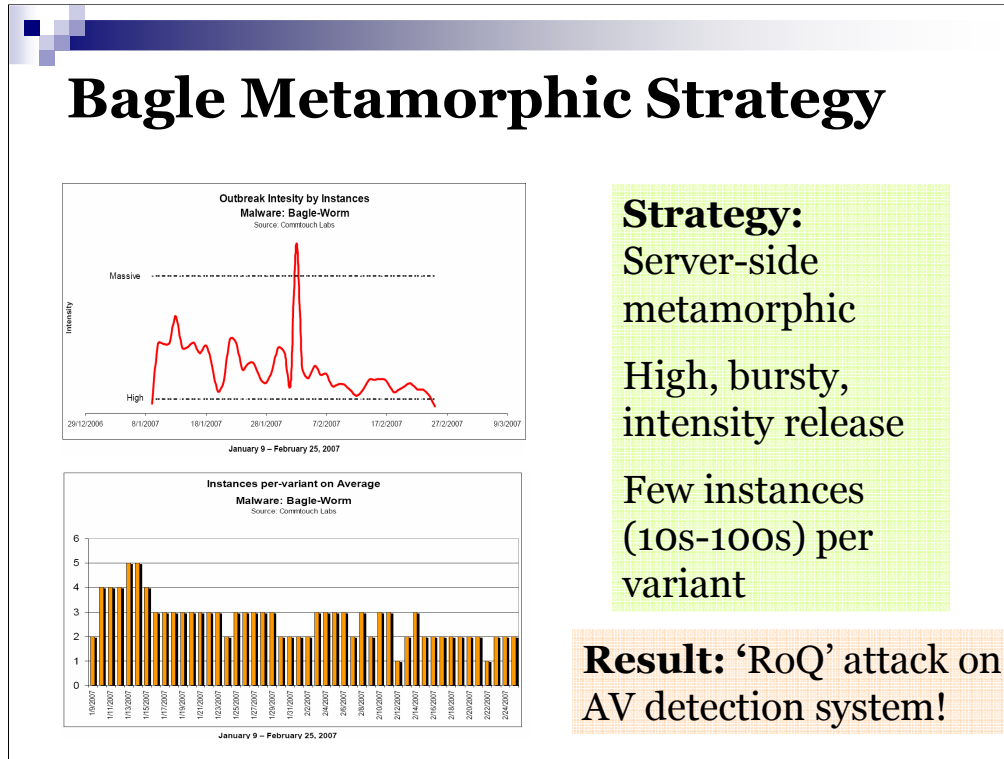
Strategy:

Server-side metamorphic
High, at times bursty, intensity release
Few instances (10s-100s) per variant

Distinct variants **since Jan 9 07:** ~30,000
Average distinct variants per day: 625
Not the only one: Feeps, Warezov ..

From Commtouch's Bagle Report: http://commtouch.com/downloads/Bagle-Worm_MOTR.pdf

The server-side polymorphic technique of writing and releasing large numbers of variants, each variant distributed via just a few email messages, is used by the malware writers to enable them to continue to defeat traditional AV solutions that are based on signatures or heuristic rules. These common anti-virus techniques depend on prior knowledge of malware to devise tools to block future outbreaks. Since server-side polymorphs like Bagle distribute each variant in a small number of emails and then switch to new variants, by the time traditional AV vendors can develop a signature or heuristic appropriate for one variant its lifecycle has ended and new variants are being propagated. Overwhelmed with a constant barrage of new variants, traditional AV solutions have difficulty keeping up”



From Commtouch's Bagle Report:

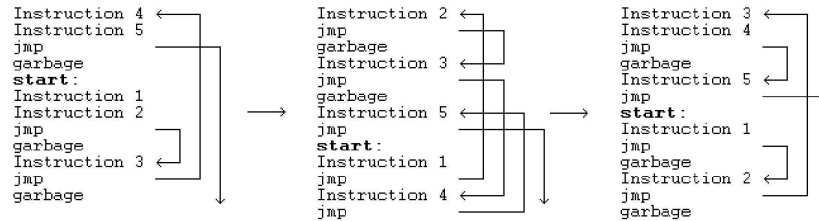
The upper graph shows intensity, the lower graph illustrates the average number of instances, or copies of the same code, per variant each day of the report period.

The data lead to the following key conclusions: Bagle-Worm writers deliberately and consistently circulate low-volume variants. 'Stealth' outbreaks are engineered to stay below the radar of AV engines. By distributing each malware variant in very low volumes (up to a few hundred instances), the malware effectively evades detection by many anti-virus solutions.

From an economic and systemic point of view, this is very smart. A set of strategies to attack *the AV cost structure* is a variant of so-called Reduction-Of-Quality attacks on systems. The result is that numerous malware variants never even reach the stage of being analyzed by the AV vendors to create a signature or heuristic.

'Classic' Metamorphic Techniques

Picture from Szor



Control Flow: Instruction reordering, branch conditions reversal, JMPs inserted randomly, subroutine permutations

Instruction: Garbage opcodes in unreachable code areas, opcode substitution with semantic equivalents (e.g. SUB EBX, EBX into XOR EBX, EBX), register swapping

Advanced Metamorphic Techniques

Code Integration: Malware merges into benign host code. “Islands” of MW code positioned into random locations and linked by jumps. No guarantee MW entry point will be reached

To integrate itself, MW **disassembles & rebuilds** host
Non-trivial: Must iteratively recompute relative addresses after insertions.

[Requires 32MB RAM, explicit section names (DATA, CODE, etc.) in host binary]



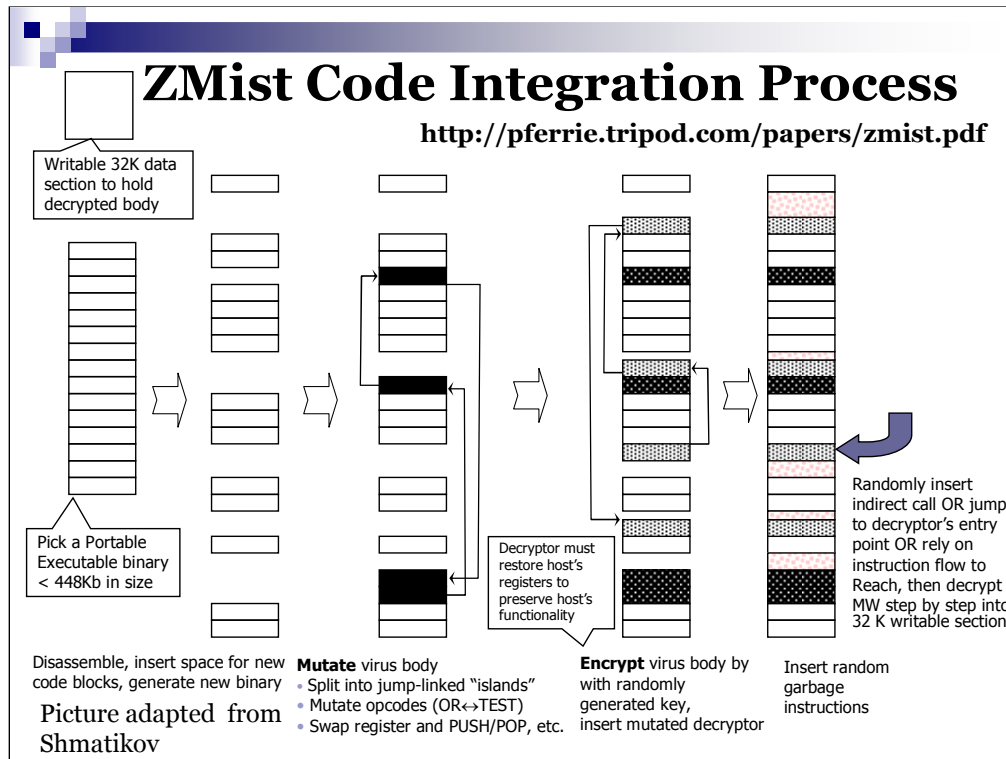
T-1000 from “Terminator 2”

Examples: ZMist and Simile

See an overview of the Mistfall engine used in ZMist at <http://vx.netlux.org/lib/vzo21.html>

The whole procedure is non trivial

- 1) Addresses are based on offsets, which must be recomputed when new instructions are inserted
- 2) MW must perform complete instruction-by-instruction disassembly and re-generation of the host binary. This is an iterative process: rebuild with new addresses, see if branch destinations changed, then rebuild again
- 3) Host binary requirements: 32MB of RAM and explicit section names (DATA, CODE, etc.) in the host binary, so it does not work with every host binary.



From <http://pferrie.tripod.com/papers/zmist.pdf>

The polymorphic decryptor consists of islands of code that are integrated into random locations throughout the host code section and linked together by jumps. The decryptor integration is performed in the same way as for the virus body integration. Existing instructions are moved to either side, and a block of code is placed in between

them. The polymorphic decryptor uses absolute references to the data section, but the Mistfall engine will update the relocation information for these references too.

An anti-heuristic trick is used for decrypting the virus code: instead of making the section writable in order to alter its code directly, the host is required to have, as one of the first three sections, a section containing writable, initialized data. The virtual size of this section is increased by 32 KB, large enough for the decrypted body and all the variables used during decryption. This allows the virus to decrypt code directly into the data section, and transfer control to there. If such a section cannot be found, then the virus will infect the file without using encryption. The decryptor will receive control in one of four ways: via an absolute indirect call (0xFF 0x15), a relative call (0xE8), a relative jump (0xE9), or as part of the instruction flow itself. If one of the first three methods is used, the transfer of control will usually appear soon after the entry point. In the case of the last method, though, an island of the decryptor is simply inserted into the middle of a subroutine, somewhere in the code (including before the entry point).

All used registers are preserved before decryption and restored afterwards, so the original code will behave as before. Zombie calls this last method .UEP., perhaps an acronym for Unknown Entry Point, because there is no direct pointer anywhere in the file to the decryptor.

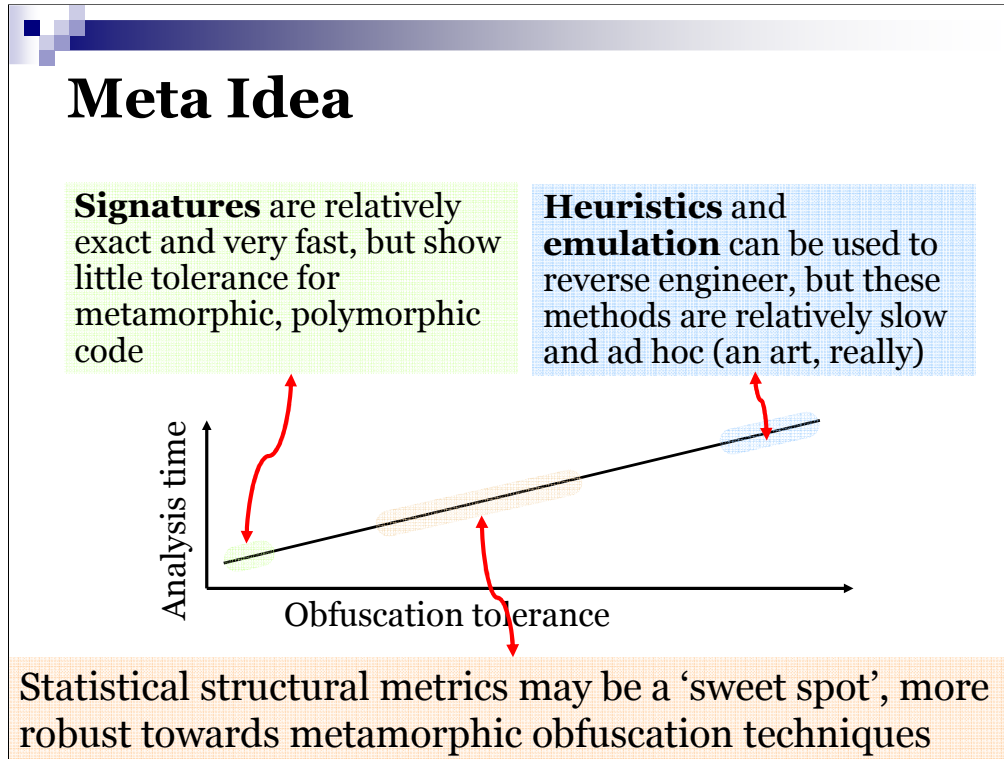
When encryption is used, the code is encrypted with ADD/SUB/XOR with a random key, and this key is altered on each iteration by ADD/SUB/XOR with a second random key. In between the decryption instructions are various garbage instructions, using a random number of registers, and a random choice of loop instruction, all produced by the Executable Trash Generator engine (ETG), also written by Zombie. It is clear that randomness features very heavily in this MW.

Approach: Structural Analysis

Goal: Identify (and classify) modern malware quicker

Problem: Signature matching and checksums tend to be too rigid, heuristics and emulation may take too long a time

Approach: Find classifiers ('structural fingerprints') that are **statistical** in nature, **'fuzzier' metrics** between static signatures and dynamic emulation and heuristics



Want analysis speed of signature with tolerance of obfuscation for heuristics/emulation

Most statistical metrics do not care about permutations. Trivial example is statistical averaging: $\text{ave}(5,2,1) = \text{ave}(1,2,5) = 2.67 \rightarrow$ idea is to find metrics that are more resilient to obfuscated code by tolerating of permutations etc

Structural Perspectives

Research 2005-2007: Tried several approaches for statistical structural discrimination

Structural Perspective	Description	Statistical Fingerprint	static / dynamic?
Assembly instruction	Count different instructions	Opcode frequency distribution	Primarily static
Win 32 API call	Observe API calls made	API call vector	Primarily dynamic
Callgraph	Explore inter- and intra-control flow of functions	Graph structural properties	Primarily static

static = read malware binaries

dynamic = execute malware binaries

1st Fingerprint: Win 32 API Calls

Joint work with Chris Ries (honors student)

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

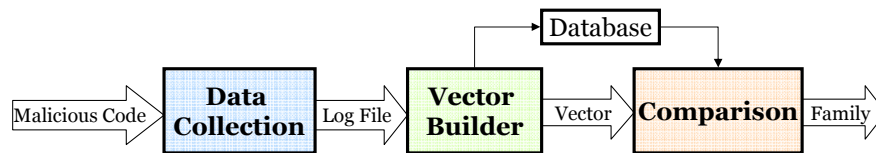
Goal: Classify malware quickly into a **family** (set of variants make up a family)

Main result (2005) : Simple VSM model yields > 80% correct classification (after tuning)

Classification purposes

Chris Ries (2005, CS honor's thesis, Colby College), now at VigilantMinds

Win 32 API calls: Overview



Data Collection: Run malicious code, recording Win32 API calls it makes

Vector Builder: Build count vector from collected API call data and store in database

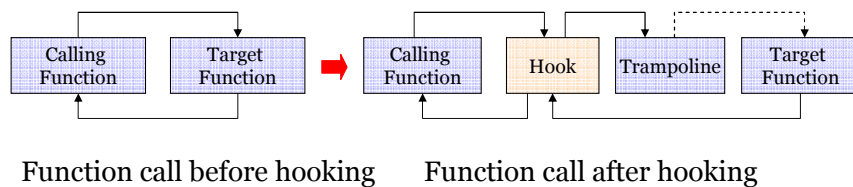
Comparison: Compare vector to all other vectors in the database to see if its related to any of them

Win 32 API Call: Call Recording

Malicious process is started in suspended state
DLL is injected into process's address space

When DLL's `DllMain()` function is executed, it hooks the Win32 API function

Hook records the call's time and arguments, calls the target, records the return value, and then returns the target's return value to the calling function.



Slide from Chris Ries

The method that the DLL uses to hook the function was developed for Microsoft's Detours Research project .

For trampoline, see Ivanov, "API hooking revealed",
<http://www.codeproject.com/system/hooksys.asp>

Win 32 API call: Call Vector

Function Name	FindClose()	FindFirstFileA()	CloseHandle()	EndPath()	...
Number of Calls	62	12	156	0	...

Column of the vector represents a hooked function and count (i.e number of times called)

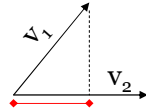
1200+ different functions recorded during execution

For each malware specimen, vector values recorded to database

Number of times a function is called can vary during execution. Run all malware in same environment to produce similar execution paths. Use comparison system that will try to account for this

Win 32 API call: Comparison

Computes cosine similarity measure csm between vector and each vector in the database

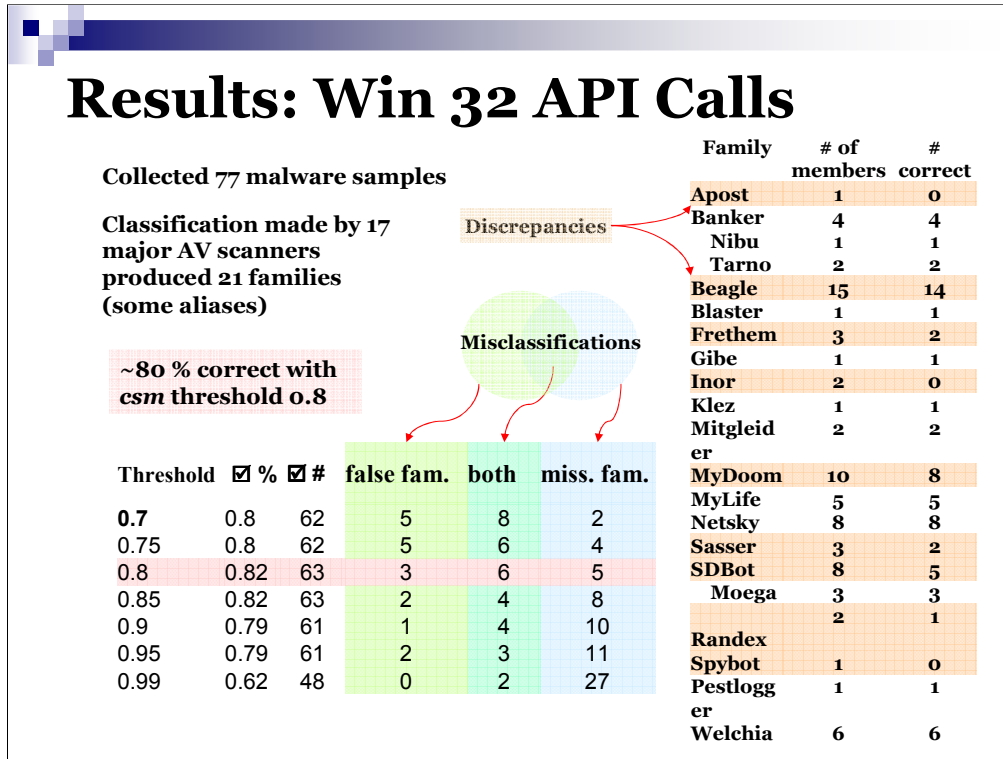


$$csm(\vec{v}_1, \vec{v}_2) = \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1| |\vec{v}_2|} = \text{red arrow}$$

If $csm(\text{vector}, \text{most similar vector in the database}) > \text{threshold} \rightarrow$ vector is classified as member of family_{most-similar-vector}

Otherwise vector classified as member of family_{no-variants-yet}

Maintains a database of all vectors that it has already processed. When a vector is inputted into the system, it is compared to every vector in the database in order to find the most similar one



2nd Fingerprint: Opcode Frequency Distribution

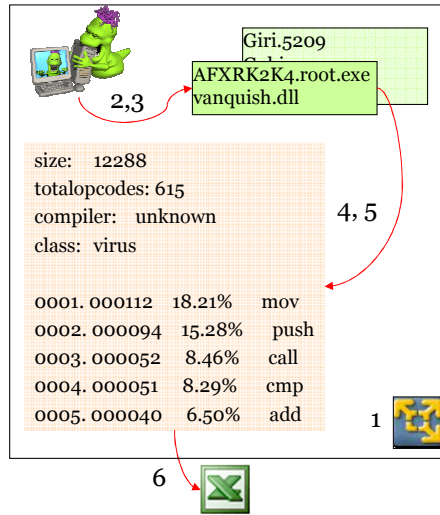
Synopsis: Statically disassemble the binary, tabulate the opcode frequencies and construct a statistical fingerprint with a subset of said opcodes.

Goal: Compare opcode fingerprint across non-malicious software and malware classes for quick identification (and later classification) purposes.

Main result (ICGeS, 2006): ‘Rare’ opcodes explain more data variation than common ones

Identification purposes

Opcode Distribution



Procedure

1. Booted VMPlayer with XP image
2. Inventoried PEs from C. Ries malware collection with Advanced Disk Catalog
3. Fixed 7 classes (e.g. virus,, rootkit, etc), chose random PEs samples with MS Excel and Index your Files
4. Ran IDA with modified InstructionCounter plugin on sample PEs
5. Augmented IDA output files with PEID results (compiler, packer) and 'class'
6. Wrote Java parser for raw data files and fed JAMA'ed matrix into Excel for analysis

Parser written in Java with JAMA matrix packages, also constructed opcode tables

Rare 14 Opcodes (parts per million)

Opcode	Goodware	Kernel RK	User RK	Tools	Bot	Trojan	Virus	Worms
bt	30	0	34	47	70	83	0	118
fdivp	37	0	0	35	52	52	0	59
fild	357	0	45	0	133	115	0	438
fstew	11	0	0	0	22	21	0	12
imul	1182	1629	1849	708	726	406	755	1126
int	25	4028	981	921	0	0	108	0
nop	216	136	101	71	7	42	647	83
pushf	116	0	11	59	0	0	54	12
rdtsc	12	0	0	0	11	0	108	0
sbb	1078	588	1330	1523	431	458	1133	782
setb	6	0	68	12	22	52	0	24
setle	20	0	0	0	0	21	0	0
shld	22	0	45	35	4	0	54	24
std	20	272	56	35	48	31	0	95

Comparison: Rare Opcode Frequencies

Opcode	Goodware
bt	30
fdivp	37
fld	357
fstcw	11
imul	1182
int	25
nop	216
pushf	116
rdtsc	12
sbb	1078
setb	6
setle	20
shld	22
std	20

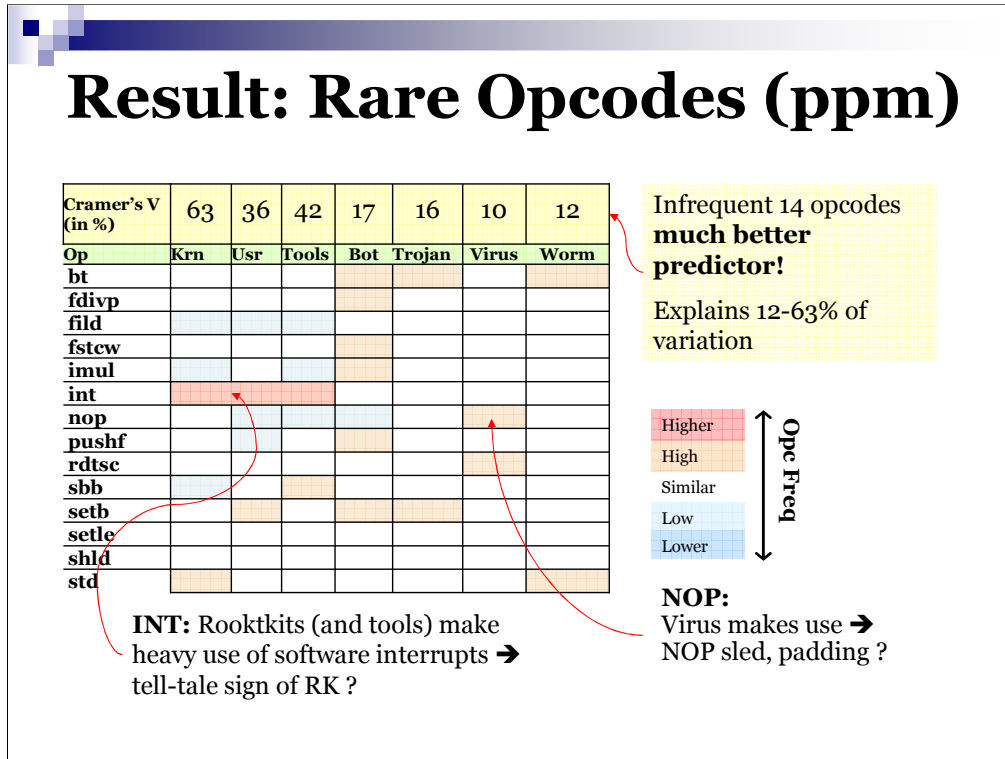
Performed stat distribution tests on frequent (not shown) and rare (shown) opcodes across 7 MW classes: Rootkits (kernel/user), Virus /Worms, Trojan, Bots, Tools

Investigate: Which, if any, opcode frequency differ between goodware and MW classes and how strong is association between opcodes and MW classes?

Contingency tables (8 * 14): Testing association using Pearson's chi square with post-hoc Haberman (1973) STAR (adjusted residual testing of individual cells) .. For further analytical procedures see Kim (2005), tinyurl.com/nws2s.

The chi-square test provides a method for testing the association between the row and column variables in a two-way table. The null hypothesis H_0 assumes that there is no association between the variables (in other words, one variable does not vary according to the other variable), while the alternative hypothesis H_a claims that some association does exist. The alternative hypothesis does not specify the type of association, so close attention to the data is required to interpret the information provided by the test.

The chi-square test is based on a test statistic that measures the divergence of the observed data from the values that would be expected under the null hypothesis of no association. This requires calculation of the expected values based on the data. The expected value for each cell in a two-way table is equal to $(\text{row total} * \text{column total}) / n$, where n is the total number of observations included in the table.



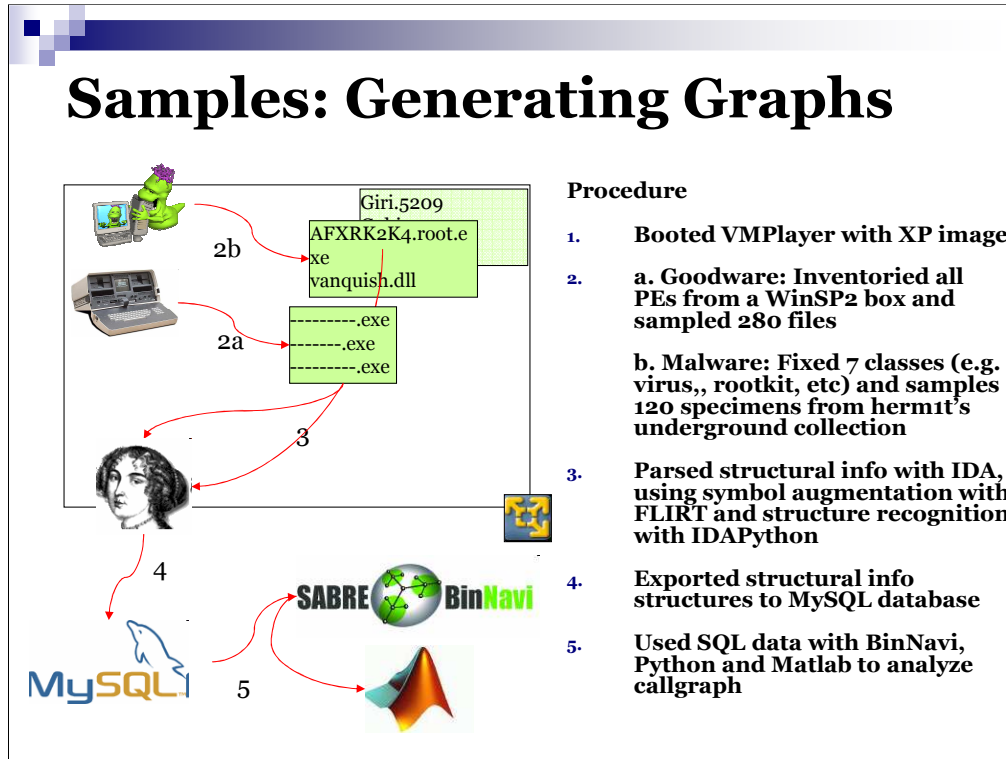
Cramer's V = strength of association between two variables

3rd Fingerprint: Graph Structure

Synopsis: Represent executables as callgraph, extract 'graph structural' fingerprints for software classes

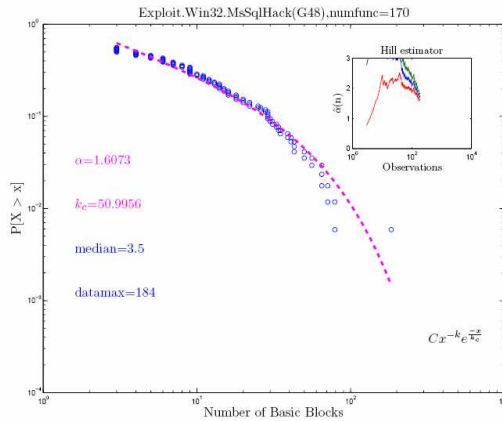
Goal: Compare 'graph structure' fingerprint of unknown binaries across non-malicious software and malware classes

Main result (AICOM, 2007): Malware tends to have a lower basic block count, implying a simpler structure: Less interaction, fewer branches, limited functionality



Thomas Dullien got HGI prize in 2006, used his software, BinNavi for this

Results: α -Fitting Pareto with Hill Check



(b) MW sample: Fitting α_{bb} and k_c

Pareto fitting ECCDFs, shown with Hill estimator inset

Investigate whether discrete distributions of $d^+(f)$; $d^-(f)$ and $d_{bb}(f)$ follows a truncated power law (Pareto)

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

k_c is cutoff point

α is slope of power law

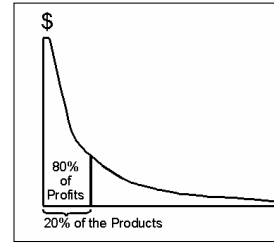
Hill estimator used for consistency check

Pareto distribution

The 80-20 rule (noted by Pareto studying Italian incomes in 1896)

Example: X is a random variable representing the income of a person in some country, and that this variable follows a power law distribution with parameters $k = \$1000$ and $a = 2$.

By the complimentary cdf (ccdf)
 1/100 of the people earn $\geq \$10,000$
 1/10,000 of the people earn \geq
 \$100,000
 1/1,000,000 of the people earn \geq
 \$1,000,000



$$f(x) = \alpha k^\alpha x^{-\alpha-1}$$

pdf

$k > 0$ is *location* parameter

$\alpha > 0$ is *slope* parameter

$$\Pr(X \geq x) = \left(\frac{k}{x}\right)^\alpha$$

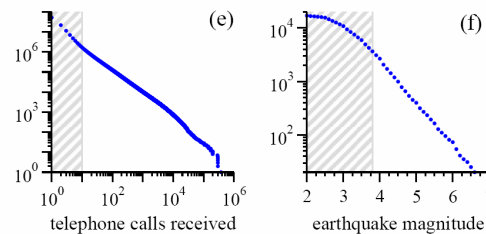
1 - cdf

See <http://arxiv.org/abs/cond-mat/0103544> for “Exponential and power-law probability distributions of wealth and income in the United Kingdom and the United States”

Power laws distribution in Graphs

Power law distributions abound in natural and engineered systems .. *why* is another lecture :D

A power law is a function $f(x)$ where the value y is *proportional to some power* of the input x

$$y = f(x) \sim x^{-\alpha}$$


Graphs from Newman

Properties of such “power-law” distributions are so-called “fat” or “heavy” tails

From Manning (UCSB),

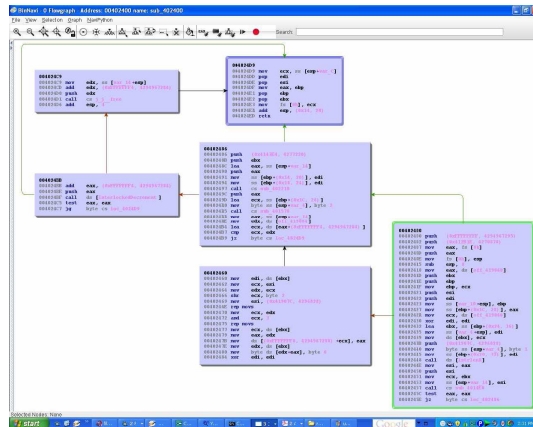
http://www.physics.ucsb.edu/~mmanning/What_are_Power_Laws.html

Why are Power Law distributions called 'Heavy-tailed'?

Many processes in nature have density functions which follow a bell-curve, or normal, or Gaussian distribution. Heights of adults pulled randomly from a human population are generally gaussian, as are sums of tosses from a weighted die. You might think that this universal bell curve must indicate universal causes -- human heights must have something in common with weighted dice. The real reason for this universality, however, is simple statistics. The Central Limit Theorem states that the sum of random variables with finite mean and finite variance will always converge to a gaussian distribution. The mean is the average value of the random variable and the variance is a measure of how much individuals differ from that average. Therefore, the gaussian is a result of universal statistical processes and not similar causes.

Interestingly, if **we relax the constraint that the variance and/or mean be finite** (in other words, we allow arbitrarily large steps and/or don't require a characteristic size or mean) then the Central Limit theorem does NOT predict Gaussians. Instead, it predicts a variety of sum-stable distributions (such as the Cauchy distribution) which all look like power laws as x becomes large. Gaussian distributions drop off quickly (large events are extremely rare), but power law distributions drop off more slowly. This means that large events (the events in the tail of the distribution) are more likely to happen in a power law distribution than in a Gaussian. This is why we call power laws heavy-tailed.

Flowgraph metrics collected



**Backdoor.Win32.Livup.c,
sub_402400**

Basic block count of a function

Instruction count in a given basic block

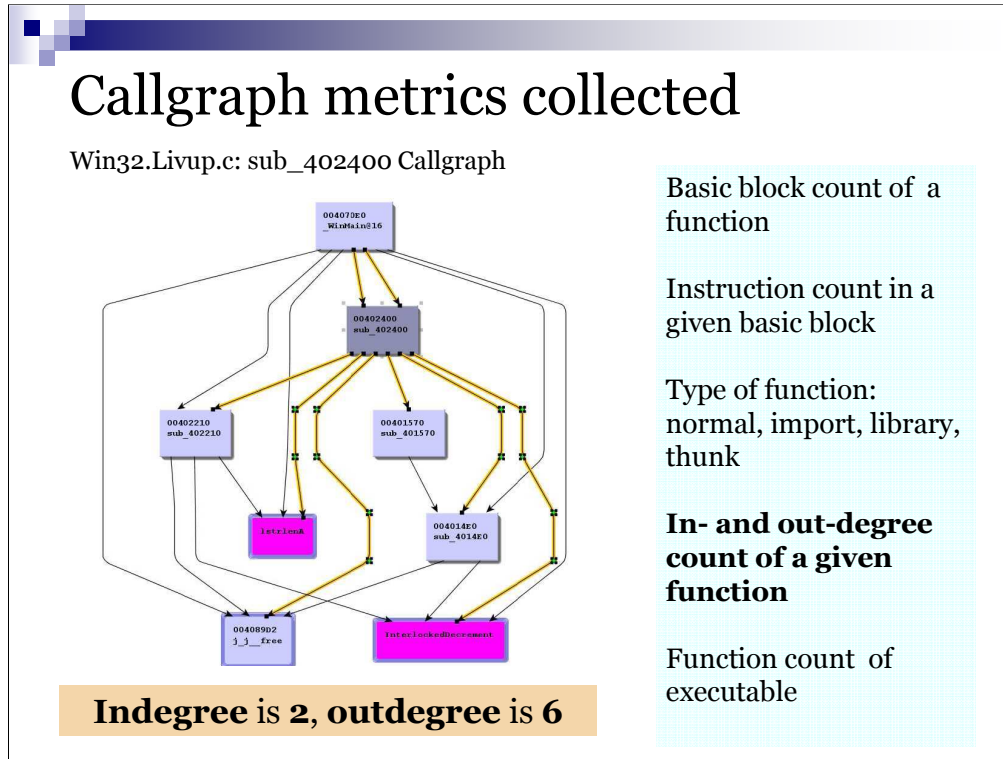
Type of function:
normal, import,
library, thunk

In- and out-degree
count of a given
function

Function count of
executable

I illustrate the concept of a control flowgraph, and basic block by means of a fragment disassembly of Backdoor.Win32.Livup.c. A control flow graph (CFG) is defined as a directed graph $G = (V, E)$ in which vertices u, v in V represent basic blocks and an edge e in E represents a possible flow of control from u to v . A basic block describes a sequence of instructions without any jumps or jump targets in the middle. I show function `sub_402400`, consisting of six basic blocks. The assembly code for one basic block starting at `0x402486` and ending with a `jz` at `0x4024B9` is given below. It consists of 16 instructions, of which two are calls to other functions. The `loc_402486` basic block is located in the middle of the flowgraph given above.

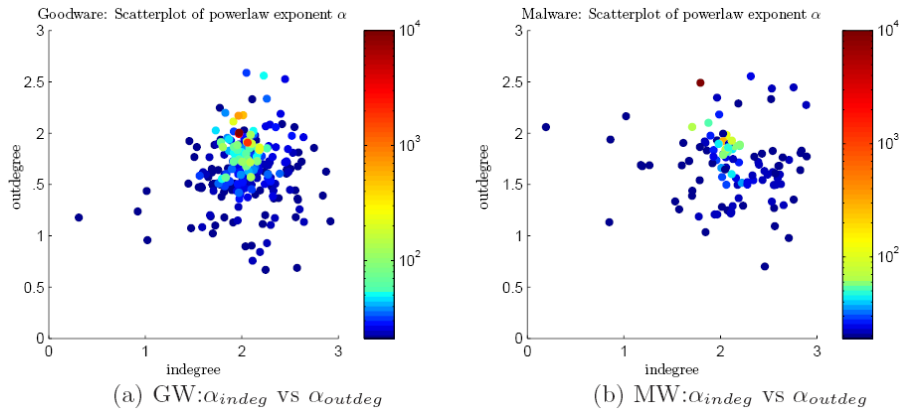
```
loc_402486:
402486 push (0x4143E4, 4277220)
40248B push ebx
40248C lea eax, ss [esp + var_14]
402490 push eax
402491 mov ss [ebp + (0x14, 20)], edi
402494 mov ss [ebp + (0x18, 24)], edi
402497 call cs sub_402210
40249C push eax
40249D lea ecx, ss [ebp + (0x1c, 28)]
4024A0 mov byte ss [esp + var_4], byte 2
4024A5 call cs sub_401570
4024AA mov eax, ss [esp + var_14]
4024AE mov edx, ds [off_419064]
4024B4 lea ecx, ds [eax + (0xF4, 429)]
4024B7 cmp ecx, edx
4024B9 jz byte cs loc_4024D9
```



I illustrate the concept of a callgraph by means of a fragment disassembly of Backdoor.Win32.Livup.c function sub_402400.

The figure shows the callgraph of function sub_402400, which indicates that sub_402400 is called twice, and in turn calls six functions.

Results: α fitting of callgraph



**$\alpha_{indeg} = [1.5-3]$, $\alpha_{outdeg} = [1.1-2.5]$ and $\alpha_{bb} = [1.1-2.1]$
with a slightly greater spread for malware**

The fitted power-law exponents α_{indeg} , α_{outdeg} , α_{bb} , together with rough callgraph size are shown above (color means the exec has more functions, e.g. dark red has $\sim 10,339$ functions)

Results: Difference Testing

Check whether there are statistical differences between (α, k_c) , the slope and the cutoff points of the distributions between goodware and malware

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

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

Table 2. α distribution fitting and testing

$\mu(k_c(bb, malware)) =$
 $\mu(k_c(bb, goodware))$
 rejected via Wilcoxon
 Rank Sum with $z = 13.4$

Only statistically relevant
 difference for **basic block**
 metrics

Byte-pattern, structural AV

Practice (bad): Classic AV metamorphic malware detection failing. Structural fingerprints probably of no help (alas ..)

Theory (worse): No algorithm can perfectly detect all possible viruses (Cohen, 1987). Even worse, MW exists which no algorithm can detect (Chess & White, 1990)

Will give you now a glimpse of novel malware, and sketch alternative detection approaches

algorithm = steps for performing closed function-based computation

Turing Machine: Assumptions

Identifies computability with the **computation of functions**. All computable problems are function based. *algorithm* = steps for performing **closed function-based computation**



Closed: All input specified at start of computation

Challenge (Knuth): Make algorithm for “toss lightly until the mixture is crumbly”

From Goldin (2005) “Breaking the Myth”:

Church-Turing Thesis: Whenever there is an effective method (algorithm) for obtaining the values of a mathematical function, the function can be computed by a TM. TMs are identified with the notion of effectiveness; computability is the current term for the same notion.

The Church-Turing thesis has since been reinterpreted to imply that Turing Machines model all computation, rather than just functions

This claim, which we call the Strong Church-Turing Thesis, is part of the mainstream theory of computation. In particular, it can be found in today's popular undergraduate theory textbooks, aka the **Strong Church-Turing Thesis**: A TM can do (compute) anything that a computer can do. We find this pars pro toto said by Sipser: “**A TM can do anything that a computer can do.**” (Michael Sipser, Introduction to the Theory of Computation, 1997)

Knuth’s problem is not algorithmic because it is impossible for a computer to know how long to mix; this may depend on conditions such as humidity that cannot be predicted with certainty ahead of time. In the function-based mathematical worldview, all inputs must be specified at the start of the computation, preventing the kind of feedback that would be necessary to determine when it's time to stop mixing.

Novel MW: K-ary Malware

Partition functionality: k distinct parts, with each part containing merely a subset of the total instructions. **Human action** may constitute a 'part' as well

Serial k-ary: Scatter code snippets. k parts act one after the another. In the wild (approx, trivial) with $k=2$: Troj/Padodor-A. Proof-Of-Concept with $4 \leq k \leq 8$ exists, but not released

Parallel k-ary: k parts as processes execute simultaneously (can watch each other too). In the wild, found with $k=4$

Out of (Turing) Box? K-ary MW

Formalization problem: Even multi-tape TM cannot thoroughly describe k-ary MW, the interaction of its parts. Cohen model too limited.

Practical problem: How to disinfect, delete parallel processes that watch one another, regenerate w/ random names after deletion? K-ary can be **self-healing**.

Result: Existing AV techniques unable to handle detection within real-time, practical constraints

From Filiol (2007):

This framework is based on vector Boolean functions instead of Turing machines as in the Cohen's model. The essential reason for that choice comes from the fact that Turing machines cannot thoroughly describe the interaction between programs, even by considering multi-tape Turing machines. Beside the fact that it would be far too complex to consider them as formalisation tools, it has been shown that the generalization of the Cohen's model is too limited.

Existing antivirus technologies are totally inefficient at detecting these k-ary codes as our study and experiments have confirmed. It would be wrong to imagine that any technical protection against these codes is tractable due to the average complexity of their detection. Detection has to face combinatorial problems that cannot be solved in an amount of time that is compatible with commercial AV

Quo vadis, AV?

Metamorphic malware

Practical detection failing
(techniques, strategy)

K-ary malware

Practical very bad, may
be out of TM 'box'?

Conjecture

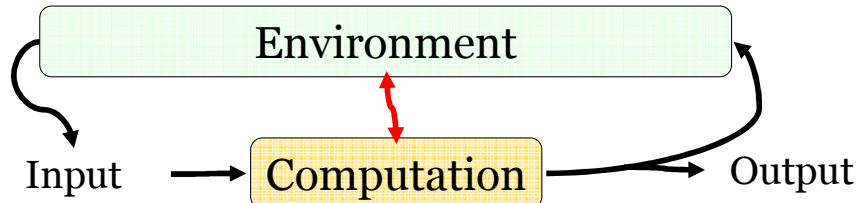
Approaches based on classic TM (Church-Turing
thesis = computation as functions) may be
insufficiently expressive to address these MW types

→ posit need for shift to 'interactive computation
model', both in practice and theory

Thought to take away: Emphasis on interaction

Interactive Computation (1997)

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



Open: I/O happens **during** computation, not just before or after (think control systems, OS, GUI). **Continuous interaction with environment**

Anticipated by Turing (1936): **Turing choice machines**, interactive choice machines as another model of computation distinct from TMs and not reducible to it

[Alan Turing](#) (1936), "On Computable Numbers, With an Application to the Entscheidungsproblem" Turing choice machines, interactive choice machines as another model of computation distinct from TMs and not reducible to it.

Goldin (2004): Persistent Turing Machines as a Model of Interactive Computation.

PTMs are provably more expressive than TM.

Why Does This Matter? AV Detection

Byte sequence-matching AV scanner based on classic TM function assumption:

Input (*data*) -> function (*decision*) -> output (*yes/no MW*)

Conjecture: Detection of metamorphic and k-ary MW via TM-based techniques likely a practical/theoretical dead-end

Heretic Conjecture: Interactive models more expressive than TMs. TM cannot compute all problems, nor do everything real computers do

Sig based AV is function-based approach.

Goldin (2005): “The Church-Turing Thesis: Breaking the Myth”

Claim 1. (Mathematical worldview) All computable problems are function-based.

Claim 2. (Focus on algorithms) All computable problems can be described by an algorithm.

Claim 3. (Practical approach) Algorithms are what computers do.

Furthermore, we looked at two more claims that have been used to corroborate the Turing Thesis myth:

Claim 4. (Nature of computers) TMs serve as a general model for computers.

Claim 5. (Universality corollary) TMs can simulate any computer.

For each of these claims, there is a grain of truth. By reformulating them to bring out the hidden assumptions, misunderstandings are removed. The following versions of the above statements are correct:

Corrected Claim 1. All algorithmic problems are function-based.

Corrected Claim 2. All function-based problems can be described by an algorithm.

Corrected Claim 3. Algorithms are what early computers used to do.

Corrected Claim 4. TMs serve as a general model for early computers.

Corrected Claim 5. TMs can simulate any algorithmic computing device.

Furthermore, the following claim is also correct:

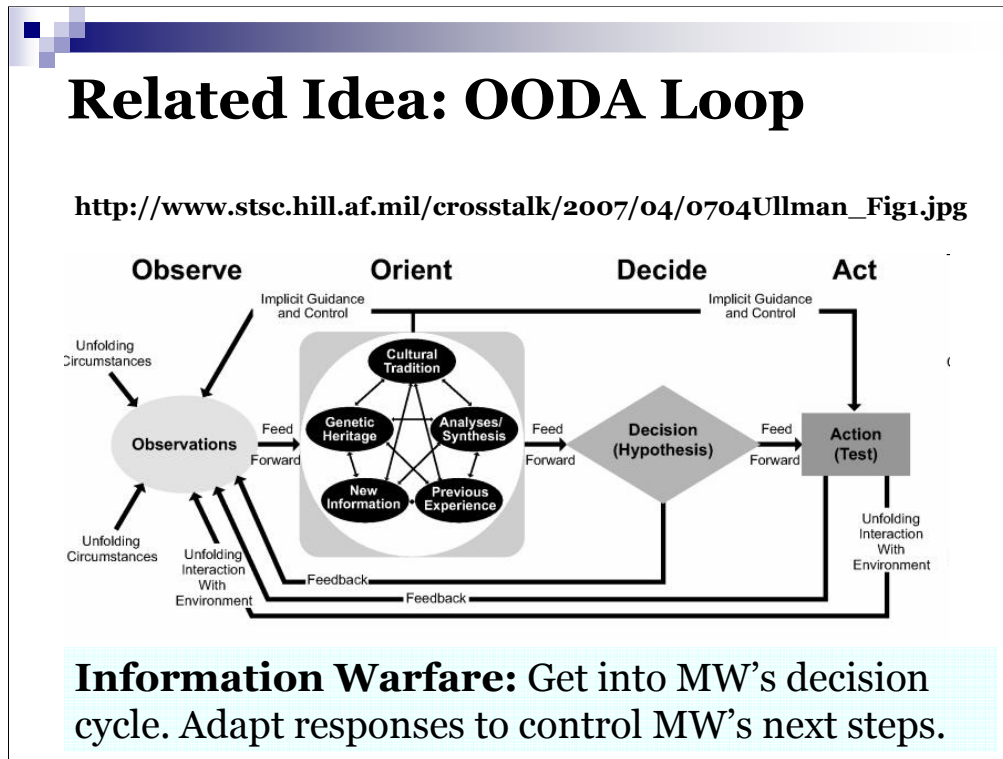
Claim 6: TMs cannot compute all problems, nor can they do everything that real computers can do.

Information-Gain Adversarial MW

Why is detection so hard, are detection rates so bad? Modern MW reduces relative information gain of AV scanning techniques

Design philosophy is a **systematic thwarting of AV information gain** by presenting a metamorphic, multi-stage, encrypted, entry-point obscuring structure

Defend with “Matrix Judo”: Adapt environment /defenses to control MW’s information gain for benefit of defender



From http://www.valuebasedmanagement.net/methods_boyd_ooda_loop.html:

The OODA loop (Observe, Orient, Decide, and Act) is an information strategy concept for information warfare developed by Colonel John Boyd (1927-1997). Although the OODA model was clearly created for military purposes, elements of the same theory can also be applied to business strategy. Boyd developed the theory based on his earlier experience as a fighter pilot and work on energy maneuverability. He initially used it to explain victory in air-to-air combat, but in the last years of his career he expanded his OODA loop theory into a grand strategy that would defeat an enemy strategically by “psychological” paralysis.

Boyd emphasized that strategy should always revolve around changing the enemy's behavior, not annihilating his forces. The parallel between Boyd's ideas and Sun Tzu's masterpiece, “The Art of War,” are obvious. Both Boyd and Sun Tzu advocate the ideas of harmony, deception, swiftness and fluidity of action, surprise, shock, and attacking the enemy's strategy.

Colonel Boyd viewed the enemy (and ourselves) as a system that is acting through a decision making process based on observations of the world around it. The enemy will observe unfolding circumstances and gather outside information in order to orient the system to perceived threats. Boyd states that the orientation phase of the loop is the most important step, because if the enemy perceives the wrong threats, or misunderstands what is happening in the environment around him, then he will orient his thinking (and forces) in wrong directions and ultimately make incorrect decisions. Boyd said that this cycle of decision-making could operate at different speeds for the enemy and your own organization. The goal should be to complete your OODA loop process at a faster tempo than the enemy's, and to take action to lengthen the enemy's loop. One tries to conduct many more loops “inside” the enemies OODA loop, causing the enemy to be unable to react to anything that is happening to him.

Colonel Boyd stated that the the enemy's OODA loop can be lengthened through a variety of means. Boyd's aim is to generate “non-cooperate” centers of gravity for the enemy through ambiguity, deception, novel circumstances, fast transient maneuvers, and the use of Sun-Tzu's idea of Cheng and Ch'i. By isolating the enemy's centers of gravity and developing mistrust and cohesion within the system (making them “non-cooperative”), friction will be greatly increased, paralysis in the system will set in, and the enemy will ultimately collapse. By attacking the thought process of the enemy / competitor, his morale and decision process can be shattered.

Passive Defenses

Prevent MW from identifying suitable target by introducing irregularities, decoys (i.e randomness) into environment

Network/Host level: Honeypots and honeynets (simulated decoys that detract from 'real' networks, hosts and services)

Program/OS level: Hot-patching binary, Address Space Layout Randomization (random heap, stack, library positioning)

For Network/Host level, see Honeynet project at <http://www.honeynet.org/>

For Program/OS level, see Shacham, H., Page, M., Pfaff, B., Goh, E., Modadugu, N., and Boneh, D. 2004. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*
<http://doi.acm.org/10.1145/1030083.1030124>

Active Defense Framework

Seeks to model MW's internal hypothesis structure, enter OODA loop, then control MW's decision/view of the world

Observation Framework: Infer MW's internal hypothesis structure via dynamic 'black-box' interaction (not through 'white-box' scanning)

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

OODA: Deception, swiftness, fluidity of action

Observation Framework: PQS

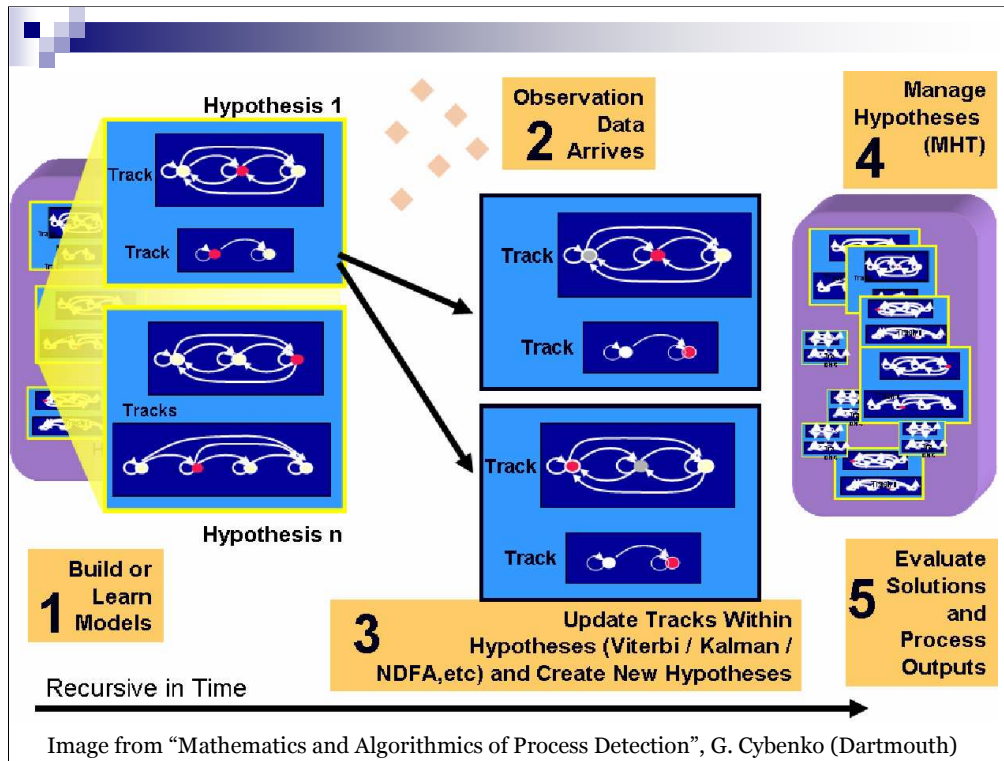
PQS: A DBMS framework that allows for **process description queries** against internal models (FSM, Markov, etc)

Components: 1) Incoming observation, 2) multiple hypothesis generation, 3) hypothesis evaluation by models, 4) model selection

MW: Observe/entice MW actions. Have PQS model MW internal decision structure

Process Query System PQSs were initially designed to solve the Discrete Source Separation Problem by setting up a DBMS framework that allows for *process description* queries against internal models, a task for which traditional DBMS are unsuitable, since the queries (e.g. SQL) are typically formulated as Boolean expressions. These models can take the form of FSMs, rule sets, Hidden Markov models, Hidden Petri Nets, and more.

The overarching goal is to detect processes by leveraging the correlation between events (such as observations) and the processes' states. PQS serves as the dynamic modeling framework of the malware. The necessary observation events are passively recorded and actively enticed through iterative interactions.



Processes have hidden states which emit observables. The relationship between observables and states is not bijective, meaning a given observation may be emitted by more than one state. The so-called 'tracks' are associations of observations to processes. Hypotheses represent consistent tracks that explain the observables.

The hypotheses in our domain correspond to the malware's internal control structure, which is inferred from its behaviour through observation. We propose that a PQS serve to dynamically 'black-box model' modern malware. The necessary observation events can be both passively recorded and actively enticed through iterative interactions.

Control Framework: Games

Game Theory: Interactive framework to weaken MW's useful and strengthen MW's useless information gain

Zero-sum: Only Win-Lose possible

Non-zero-sum: Win-Win possible

Imperfect: Hidden information

Iterative: Play many times

MW: Following PQS model of MW, want to simulate MW 'win' while MW actually 'loses'

Iterative imperfect non-zero-sum games The goal of the control framework is to create the illusion of win-win (non-zero-sum) viz the malware's goals by iteratively either weakening useful/accurate and strengthening useless/misleading information gain through defensive strategies. Game theory provides a suitable interactive framework.

History of Game Theory

folk wisdom the Holy Bible, Talmud
combinatorial games Pascal, Bernoulli (16th century)

1913 Ernst Zermelo
chess as a zero sum game

1921 Emile Borel minmax games

1928 John von Neumann minmax theorem

1942 the Michael Curtiz film Casablanca
an example of real life games

1944 John von Neumann & Oscar Morgenstern
Theory of Games and Economic Behavior

1950 John Nash introduces Nash equilibrium concept

1953 Lloyd Shapley introduces Shapley value for cooperative games

1953 prisoner's dilemma game
Harold W. Kuhn & Alan W. Tucker

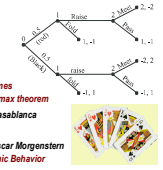
Nobel laureates

John Nash
Nash equilibrium

John Harsanyi
incomplete information, Bayesian games, 1967

Reinhard Selten
dynamic games
subgame-perfect equilibrium, 1965

Robert Auman, Thomas Schelling
Cooperative games
infinitely repeated games, 2005



Nash equilibrium
players 1 and 2,
actions x, y and
profits $\pi_1(x, y), \pi_2(x, y)$

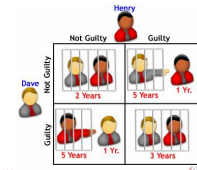
reaction curves $R_1(y), R_2(x)$

$$\pi_1(R_1(y), y) = \max_x \pi_1(x, y)$$


$$\pi_2(x, R_2(x)) = \max_y \pi_2(x, y)$$

Nash equilibrium x^N, y^N
 $x^N = R_1(y^N), y^N = R_2(x^N)$

computation
of x^N and y^N
an adjustment
process
to reach the
equilibrium

$$\begin{cases} x_{k+1} = R_1(y_k) \\ y_{k+1} = R_2(x_{k+1}) \end{cases}$$


Incomplete information and Bayesian games
players with unforeseeable behaviour enter the scene



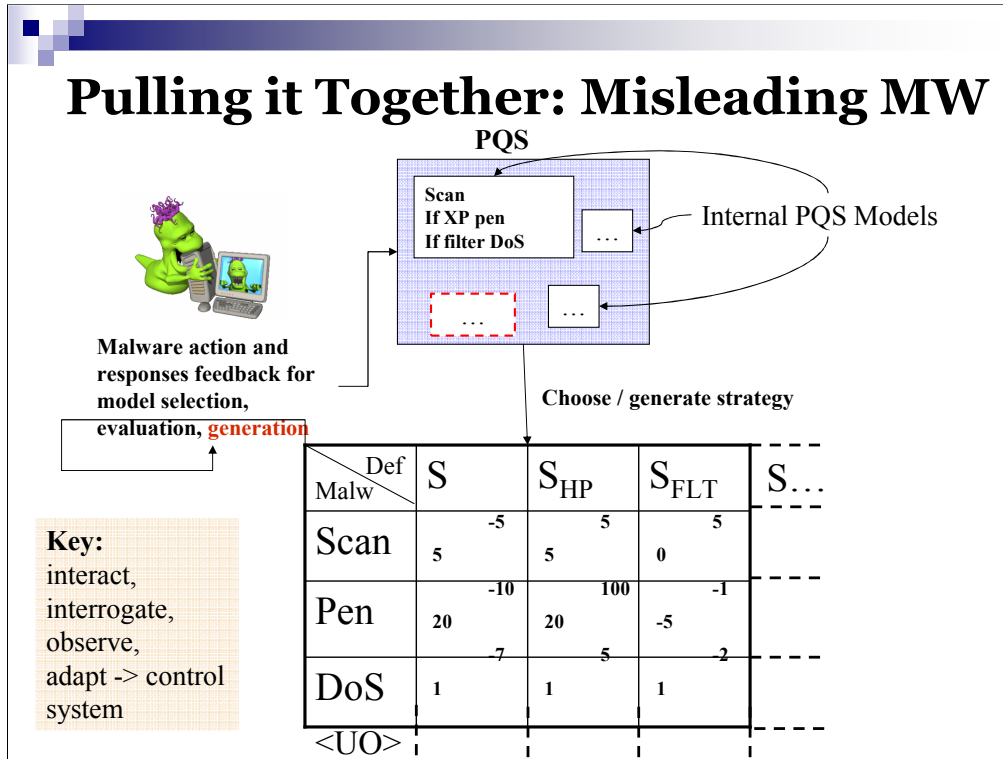
nobody knows
the other players'
true intentions,
their types...
yet, they must
play the game

Idea: Repeated, Bayesian, imperfect information game with MW

Goal: OODA-loop subverting interactive strategic 'judo' by the defense

<http://www.sal.hut.fi/Research/Posters/>

Propose interactive, OODA-loop subverting strategic 'judo' against modern malware marks a philosophical shift. From a predominantly byte sequence-matching white-box scanners premised on classic TM function-based assumptions, we suggest moving towards more 'Interactive Computation' through the use of interactive iterative games and black-box process modeling.



The feedback loop of this framework is sketched in a toy example in above figure. Suppose the malware's toy internal hypothesis structure and strategies are modeled by *Scan; if XP penetrate; if filtered DoS* in a PQS internal model.

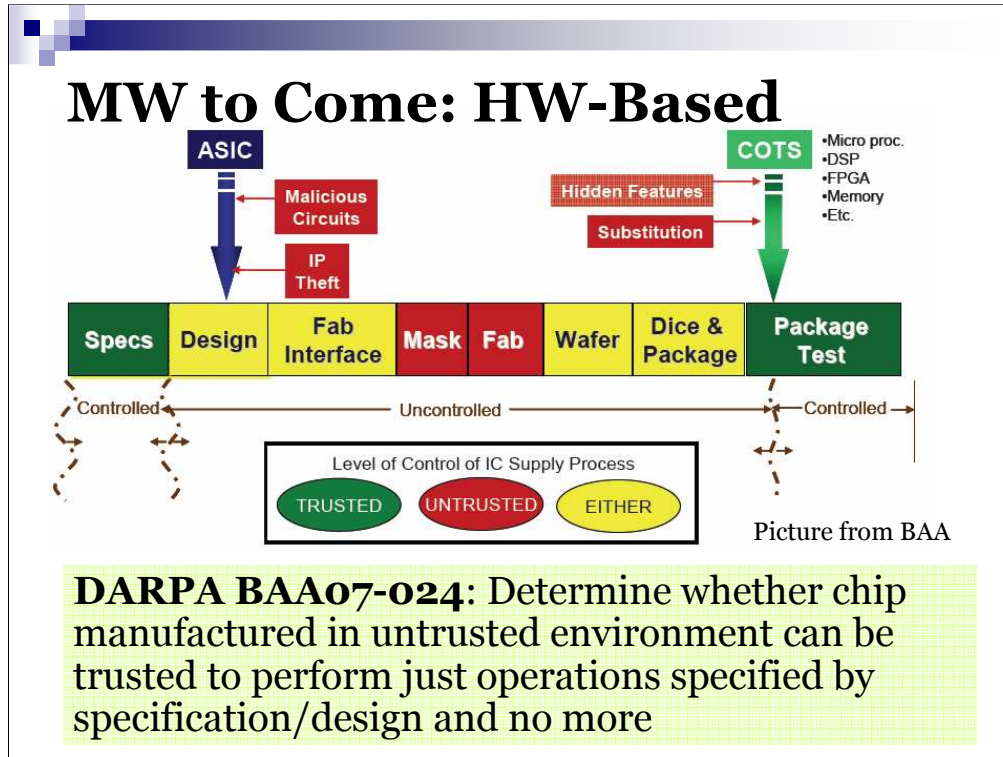
The defense's strategies are *S* (no defense), *SHNP* (honeypot), *SFLT* (filter/block ICMP response). The game matrix shows the payoffs of the defense's and malware's strategy combinations. The malware starts scanning and wants to get to [*Pen; S*] penetrating a real host. The defense wants to engage sequential strategies such that the malware penetrates a fake host [*Pen; SHP*], 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 the malware's $DKL(p(x_j H_i) || p(x_j H_j))$. Strategies may not be fixed and dynamically generated as PQS models adapt to the malware responses, as indicated by by *S...* (*new defense strategy*) and *< UO >* (Unknown observation)

MW to Come: Satan Virus

Idea: Forces humans via carrot and stick principle to do its bidding

Procedure: Give some powers (like search people's files and emails) and after some use, it blackmails user into propagating it by threatening to reveal your searches to the victim, bribes you with more access, etc.

Use humans explicitly as 'code': Entrapment through greed, malice and short-sightedness, then uses human intelligence for propagation



From Wikipedia:

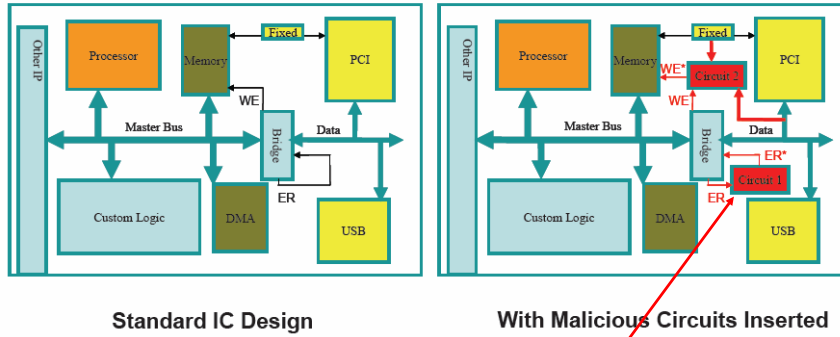
An **application-specific integrated circuit** (ASIC) is an [integrated circuit](#) (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed solely to run a [cell phone](#) is an ASIC.

[Field-programmable gate arrays](#) (FPGA) contain [programmable logic](#) components called "logic blocks", and programmable interconnects that allow the same FPGA to be used in many different applications

The general term application specific integrated circuit includes FPGAs, but most designers use ASIC only for non field programmable devices (e.g. standard cell or sea of gates) and make a distinction between ASIC and FPGAs.

FPGAs are usually slower than their [application-specific integrated circuit](#) (ASIC) counterparts, as they cannot handle as complex a design, and draw more power. But their advantages include a shorter [time to market](#), ability to re-program in the field to fix bugs, and lower [non-recurring engineering](#) costs. Vendors can sell cheaper, less flexible versions of their FPGAs which cannot be modified after the design is committed. The designs are developed on regular FPGAs and then migrated into a fixed version that more resembles an ASIC.

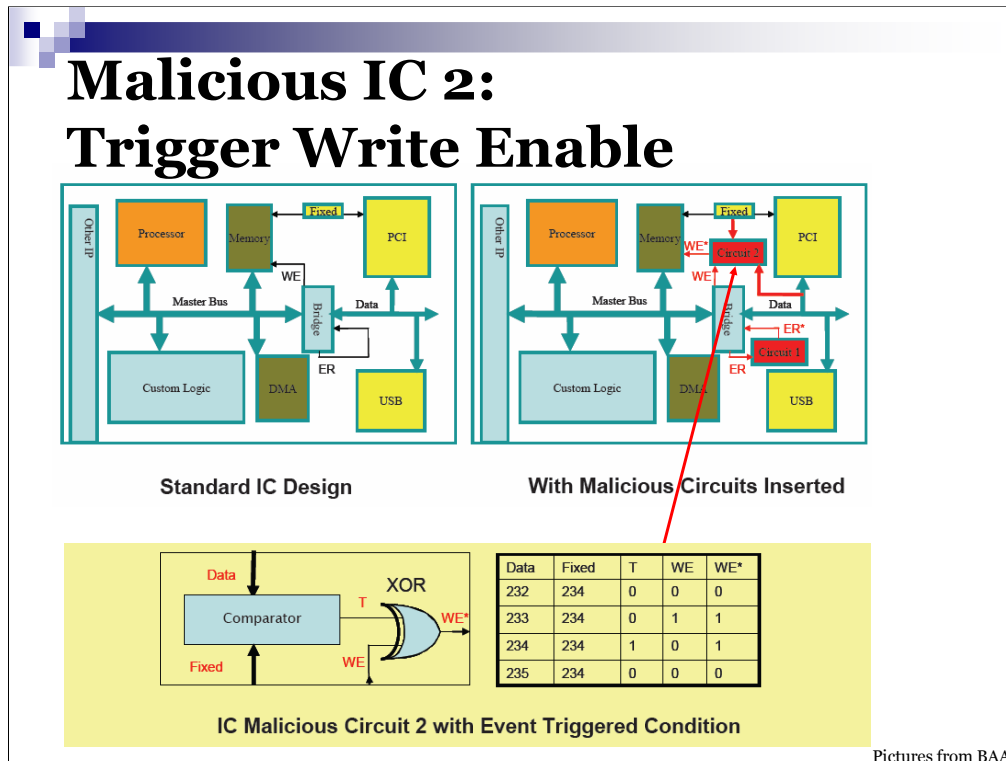
Malicious IC 1: Turn off Error Reporting



T	ER	ER*
1	0	1
1	1	0

IC Malicious Circuit 1 with Trigger Always On Condition

Pictures from BAA



Expert opinion:

“In a corporate environment it is next to impossible to verify the integrity of an integrated circuit. Most systems employ (re-)programmable circuits or even microprocessors. Even if it should be feasible to verify the underlying logic, you would have to re-do (large) parts of the verification process in case of any update.

[..]

Especially in the case of a (purported) security fix the risk of deploying a trojanised firmware would be weighted much lower than the (possible) impact of the (purported) security vulnerability. [...]

And regarding the impact on computer forensics: Johanna Rutkowska demonstrated how to reprogram the North Bridge of an AMD platform in a way that at **the same physical address the DMA controller and the CPU access different portions of memory**. Mind you, that was not an undocumented feature or a faulty IC. It's just an example of "unexpected" or "clever" usage of documented system behavior - and an example of bad design as well.”

Quantum Cryptography

In short, quantum cryptography solves (1-time pad) **key distribution problem**

Quantum channel sensitive to Eve .. and noise!

Alice's bit sequence:	0	0	1	0	1	0	1	1	1
Alice's filter scheme:	/	/	\	/	\	/	/	\	/
Bob's detection scheme:	+	+	+	+	×	+	+	×	+
Bob's bit measurements:	1	0	1	0	1	0	0	1	1
Retained bit sequence (key):	-	0	-	0	1	-	-	1	1

Picture from Janusz Kowalik (U. Washington)

How it works:

User 1 sends photons: | | / - - \ - | - /
 User 2 sets filter: X + + X X X + X + +
 User 2 receives photons / | - \ / \ - / - |
 User 2 tells User 1 (publicly) settings
 User 1 tells User 2 (publicly) which settings correct: 2, 6, 7, 9
 Both users keep those states correctly measured:

* | * * * \ - * - *

Using { \, | } = 0 and { - , / } = 1 yields:
 0 0 1 1 : Shared Key for one time pad

Why it can detect eavesdroppers (from <http://www.csa.com/discoveryguides/crypt/overview.php>):

If an eavesdropper Eve tries to gain information about the key by intercepting the photons as they are transmitted from Alice to Bob, measuring their polarization, and then resending them so Bob does receive a message, then since Eve, like Bob, has no idea which basis Alice uses to transmit each photon, she too must choose bases at random for her measurements. If she chooses the correct basis, and then sends Bob a photon matching the one she measures, all is well. However, if she chooses the wrong basis, she will then see a photon in one of the two directions she is measuring, and send it to Bob. If Bob's basis matches Alice's (and thus is different from Eve's), he is equally likely to measure either direction for the photon. However, if Eve had not interfered, he would have been guaranteed the same measurement as Alice. In fact, in this intercept/resend scenario, Eve will corrupt 25 percent of the bits. So if Alice and Bob publicly compare some of the bits in their key that they have both correctly measured and find no discrepancies, they can conclude that

Quantum Malware?

Not speculative: Quantum Cryptography used today: Swiss national elections, Oct 21st 2007 (<http://tinyurl.com/3ctx4y>)

To come? QMW designed to decohere qbit's phase and thus randomize data through phase gates, distort operations of quantum networks by malicious interference

Would need *really* new approaches in terms of Quantum AV (metrology, error correction) to clean, restore data. Open interesting problem!

See New Scientist Article: <http://tinyurl.com/22fbcn>

A **quantum computer** is any device for [computation](#) that makes direct use of distinctively [quantum mechanical phenomena](#), such as [superposition](#) and [entanglement](#), to perform operations on data. In a classical (or conventional) computer, information is stored as [bits](#); in a quantum computer, it is stored as [qubits](#) (**quantum bits**). The basic principle of quantum computation is that the quantum properties can be used to represent and structure data, and that quantum mechanisms can be devised and built to perform [operations](#) with this data.

Researchers say the emergence of quantum malware is an inevitability, but only recently has serious debate about protecting computers from such programs started, compared to the decades of research and billions of dollars already committed to quantum computer development. Quantum computers have yet to be fully realized, but a "quantum Internet" comprised of optical fiber and free space point-to-point networks dedicated to channeling quantum information already exists. This prompted University of Toronto researchers Lian-Ao Wu and Daniel Lidar to author a 2005 paper detailing a defense against quantum malware. Lidar says a quantum communication network will invite interference like any other network, while hackers could "decohere" a quantum bit's phase information and cause the output to randomize. Wu and Lidar recommend that quantum systems be kept offline as long as possible, and they propose a back-up system in which all networked quantum computers have an ancillary register of qubits equal in size to the quantum computer's memory, which is isolated whenever the computer is linked to the network to prevent direct infection. All members of a network share a secret sequence of run-time intervals that are very brief, and that must be considerably shorter than the periods when the calculations are stored in the ancillary qubit register. The setup of quantum computer networks in which more than a few kilometers separates the computers necessitates the inclusion of "quantum repeater" boxes, which could be hijacked. Lidar suggests an alternative device he and Wu conceived that uses the most simple optical components installed at regular intervals along the optical fiber.

Synopsis/Thoughts/Speculation

Need push in theory development in context of interactive models to formalize modern software malware (maybe hardware too?)

For practical deployment, leverage 'black-box' modeling techniques and interaction frameworks. For HW-based, who knows??

Philosophically, modern malware investigation may resemble **naturalist approaches** of Alexander Humboldt/E.O. Wilson .. Could MW 'art' be imitating life along lines that make MW investigation a new subfield of natural sciences?



Advice to students (especially)

Dare to be bold, stake out a position, then argue scientifically, empirically and logically

Imagination as the first step **is much more important than knowledge**

Be wrong at times, you cannot grow if you do not take that chance

Thank you for your time and the invitation to speak at the Horst Görtz Institut 2007 at the Ruhr Universität Bochum

References

Eds: Dina Goldin, Scott A. Smolka, Peter Wegner. “Interactive Computation: The New Paradigm” (Springer, 2006)

Filiol, Eric. “Formalisation and implementation aspects of K -ary (malicious) codes”, *J Comp Vir* 3:2 (2007)

Aycock, J. , deGraaf, R; Jacobson M. “Anti-disassembly using cryptographic hash functions”, *J Comp Vir* 2:1 (2006)



References *(cont)*

Bilar, D. Opcodes as Predictor for Malware. To appear: International Journal of Electronic Security and Digital Forensics (2008)

Bond, M. and Danezis, G. A pact with the devil. In: Proc of the 2006 Workshop on New Security Paradigms (2007)

Binmore, K. "Playing for Real: A Text on Game Theory" (Oxford U Press, 2007)



References *(cont)*

Wu, Lian-Ao and Lidar Daniel. “Quantum Malware”. *Quantum Information Processing* **5**:69 (2006)

Bilar, D. “Misleading modern malware”.
Submitted: *Journal in Computer Virology* (2007)

Cybenko, G.; Berk, V.: An overview of process query systems. In: *Proc SPIE Tech Homeland Sec and Def*, Vol. 5403 (2004)

References *(cont)*

DARPA BAA07-24: “TRUST in Integrated Circuits”. <http://tinyurl.com/3y7nno> (2007)

Chess, D. and White, S.: “An Undetectable Computer Virus” <http://tinyurl.com/74vgv> (2000)

Bilar, D.: “Callgraph structure of executables”. To appear: AI Comm. Special Issue on “Network Analysis in Natural Sciences and Engineering” (2007)

References *(cont)*

Newman, M.: “The Structure and Function of Complex Networks” (2003).

<http://arxiv.org/abs/cond-mat/0303516>

Goldin, D and Wegner, P.: “The Church-Turing Thesis: Breaking the Myth”. In: Springer Lecture Notes in Computer Science, Vol. 3526 (2005)

Wegner, P.: “Why Interaction is more Powerful than Algorithms. Communications of the ACM (May 1997)