OPCODES AS PREDICTOR FOR MALWARE

Daniel Bilar
Department of Computer Science, Wellesley College, Massachusetts, USA
dbilar@wellesley.edu

Abstract: This paper discusses a detection mechanism for malicious code through statistical analysis of opcode distributions. 67 malware executables were sampled statically disassembled and their statistical opcode frequency distribution compared with the aggregate statistics of twenty non-malicious samples. We find that malware opcode distributions differ statistically significantly from non-malicious software. Furthermore, rare opcodes seem to be a stronger predictor, explaining 12-63% of frequency variation.

1. Motivation:

World-wide financial damages induced by malware passed the $US10b mark in 1999; they had been averaging around US$14b for the last seven years (Computer Economics, 2007). In the same time span, the host base (end systems with an IP address) grew from 56m to roughly 440m, according to one estimate (Zakon, 2006). Viewed in conjunction with the smaller relative growth in damages, these numbers – if roughly correct - could be interpreted as a success story for signature-based anti-viral (AV) software, which is routinely deployed on personal computers nowadays.
However, malware is evolving, and initial AV detection rates for recent modern malware do not look reassuring. In February 2007, for instance, seventeen state-of-the-art, updated AV scanners were checked against twelve well-known, previously submitted, highly polymorphic and metamorphic malware samples. The miss rate was 100% to 0%, with an average miss rate of roughly 38% (Clementi, 2007). The theoretical aspects of such metamorphic self-reproducing programs were presciently laid out 27 years ago (Kraus, 1980) and the emerging practical deployment of such malware predicted in 2001 (Szor, 2001).

2. Introduction:

The goal of this paper was to compare opcode distributions of malicious and non-malicious software and give a preliminary assessment of its usefulness for detection and differentiation of modern (polymorphic and metamorphic) malware. Polymorphic malware contain decryption routines which decrypt encrypted constant parts of the malware body. The malware can mutate its decryptors in subsequent generations, thereby complicating signature-based detection approaches. The decrypted body, however, remains constant.
Metamorphic malware generally do not use encryption, but are able to mutate their body in subsequent generation using various techniques, such as junk insertion, semantic NOPs, code transposition, equivalent instruction substitution and register reassignments (Christodorescu, 2003) (Szor, 2005, pp.256-270) .
The net result of these techniques is a continuously staler (time-sensitive) signature base suitable for pattern-based detection approaches, as recent server-side polymorphic malware proliferation amply demonstrated (Commtouch, 2007).
Since signature-based approaches are quite fast (but show little tolerance for metamorphic and polymorphic code) and heuristics such as emulation are more resilient (but quite slow and may hinge on environmental triggers), a detection approach that combines the best of both worlds would be desirable. This is the philosophy behind a *structural fingerprint.* Structural fingerprints are statistical in nature, and as such are positioned as 'fuzzier' metrics between static signatures and dynamic heuristics.
The structural fingerprint considered in this paper is based on the extended x86 IA-32 binary assembly instructions without arguments, from

Opcode	Goodware	Kernel RK	User RK	Tools	Bot	Trojan	Virus	Worms
mov	25.3%	37.0%	29.0%	25.4%	34.6%	30.5%	16.1%	22.2%
push	19.5%	15.6%	16.6%	19.0%	14.1%	15.4%	22.7%	20.7%
call	8.7%	5.5%	8.9%	8.2%	11.0%	10.0%	9.1%	8.7%
pop	6.3%	2.7%	5.1%	5.9%	6.8%	7.3%	7.0%	6.2%
cmp	5.1%	6.4%	4.9%	5.3%	3.6%	3.6%	5.9%	5.0%
jz	4.3%	3.3%	3.9%	4.3%	3.3%	3.5%	4.4%	4.0%
lea	3.9%	1.8%	3.3%	3.1%	2.6%	2.7%	5.5%	4.2%
test	3.2%	1.8%	3.2%	3.7%	2.6%	3.4%	3.1%	3.0%
jmp	3.0%	4.1%	3.8%	3.4%	3.0%	3.4%	2.7%	4.5%
add	3.0%	5.8%	3.7%	3.4%	2.5%	3.0%	3.5%	3.0%
jnz	2.6%	3.7%	3.1%	3.4%	2.2%	2.6%	3.2%	3.2%
retn	2.2%	1.7%	2.3%	2.9%	3.0%	3.2%	2.0%	2.3%
xor	1.9%	1.1%	2.3%	2.1%	3.2%	2.7%	2.1%	2.3%

Table 1: Comparison of the 14 most frequent opcodes

random software samples, blocked for criteria described below. Section 3 gives a review and an evaluation of related classification and detection research. Sections 4 and 5 outline the sampling, opcode extraction and statistical testing procedures. Sections 6, 7, 8 and 9 discuss findings, improvements to the presented approach, malware on the horizon and contributions of this research, respectively.

3. Related Work:

Explicitly statistical analysis' of structural features of binaries files were undertaken by (Li, 2005) and (Weber, 2002). Li et al used 1-gram analysis of binary byte values (not opcodes) to generate a fingerprint (a 'fileprint') for file type identification and classification purposes. Weber et all start from the assumption that compiled binaries exhibit homogeneities with respect to several structural features such as instruction frequencies, instruction patterns, memory access, jump/call distances, entropy metrics and byte-type probabilities and that tampering by malware would disturb these homogeneities. They indicated having implemented a comprehensive PE Analysis Toolkit ('PEAT') and tested it on several malware samples. Sadly, no results beyond some tantalizing morsels are given. Attempts to contact the authors for a version of PEAT were also unfortunately for naught.

Further static and dynamic malware investigations were undertaken by (Chinchani, 2005) (Rozinov, 2005) and (Polychronakis, 2006) (Ries, 2005) (Bilar, 2007) (Bayer, 2006), respectively. Chinchani et al implemented an involved scheme for statically detecting exploit code of a certain general structure (NOP sled, payload, return address) in network streams by analyzing data and control flow information. They reported robust results vis-à-vis metamorphic malware.

With an eye towards detection of self-contained polymorphic shellcode, Polychronakis et al implemented a full-blown NIDS-embedded x86 emulator that speculatively executes potential instruction sequences in the network stream to compare it against polymorphic shellcode behaviour.

Their tuned behavioural signature is partly opcode-sequence based: An execution chain containing either a `call`, `fstenv`, or `fsave` instruction, followed by a read from the memory location where the instruction pointer was stored as a result of one of the above instructions, followed by some tuned number of specific memory reads is interpreted as shellcode. They validated their nifty scheme against thousands of shellcode instances created by ten different state-of-the-art polymorphic shellcode engines, with zero false negatives.

Bayer and Ries' behavioural analysis implementations took a different approach: They

Opcode	Goodware	Kernel RK	User RK	Tools	Bot	Trojan	Virus	Worms
bt	30	0	34	47	70	83	0	118
fdvip	37	0	0	35	52	52	0	59
fild	357	0	45	0	133	115	0	438
fstcw	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

Table 2: Comparison of rare opcodes (in parts per million)

ran the malware dynamically in a sandbox, record security-relevant Win32 API calls, and constructed a syscall-based behavioural fingerprint for malware identification and classification purposes. Rozinov, on the other hand, located calls to the Win32 API in the binary itself: While Ries and Bayer recorded the malware's system calls dynamically during execution, Rozinov statically disassembled and simplified the malware binary via slicing, scanned for Win32 API calls and constructed an elaborate Finite State Automaton signature for later detection purposes.

Recently, graph-based structural approaches gained some traction. (Flake, 2005) proposed a simple but effective signature set to characterize statically disassembled binaries: Every function in the binary was characterized by a 3-tuple (number of basic blocks in the function, number of branches, and number of calls). These sets were used to compare malware variants and localize changes. (Bilar, 2007) examined the static callgraphs of 120 malicious and 280 non-malicious executables. He fitted Pareto models to the in-degree, out-degree and basic block count distributions, and found a statistically significant difference for the derived power law exponent of the basic block count fit. He concluded that malware tended to have a lower basic block count than non-malicious software, implying a simpler structure: Less interaction, fewer branches, and more limited functionality.

In an exemplary exposition for the purposes of worm detection, (Kruegel, 2006) extracted control flow graphs from executable code in network streams, augmented them with a colouring scheme, identified k-connected subgraphs that were subsequently used as structural fingerprints. He evaluated his scheme offline against 342 malware samples from 93 distinct families.

The general problem with pattern-based approaches is not accuracy; the individual classifiers can be tuned to the desired false negative or positive rates. The problem is really one of *practical detection speed*: As the adversarial dissimulation techniques of malware continue to evolve, computational complexity issues (Spinellis, 2003) will soon show the practical limits of the more involved emulation and parsing schemes. Structure-based approaches (based on opcode frequencies and callgraph structures, for instance) may capture enough semantic richness to detect dissimulated malware without the necessity of full-blown emulation.

Opcode	Kernel RK	User RK	Tools	Bot	Trojan	Virus	Worms
mov	36.8	20.6	2.0	70.1	28.7	-27.9	-20.1
push	-15.5	-21.0	4.6	-59.9	-31.2	12.1	6.9
call	-17.0	1.2	5.2	26.0	10.6	2.6	-0.3
pop	-22.0	-13.5	4.9	5.1	9.8	4.8	-1.1
cmp	7.4	-3.5	-0.6	-30.8	-21.2	4.7	-1.8
jz	-7.4	-6.1	0.9	-20.9	-11.0	1.4	-4.4
lea	-16.2	-8.4	10.9	-29.2	-18.3	11.5	4.2
test	-12.2	0.0	-6.6	-14.6	1.8	-0.2	-3.4
jmp	8.5	11.7	-5.0	-2.2	5.0	-2.3	20.4
add	22.9	10.8	-6.4	-13.5	-0.1	4.3	0.5
jnz	8.7	7.4	-11.7	-12.2	-0.9	5.3	8.0
retn	-5.5	2.5	-12.3	18.4	17.8	-1.4	2.6
xor	-8.9	6.7	-2.6	29.5	15.3	2.7	7.7
and	1.9	-7.3	-0.7	-33.6	-17.0	2.4	5.9

Opcode Frequency

Higher
High
Similar
Low
Lower

Tests suggests opcode frequency roughly

1/3 same
1/3 lower
1/3 higher

vs goodware

Opcode	Kernel RK	User RK	Tools	Bot	Trojan	Virus	Worms
bt	-1.2	-0.4	0.7	6.6	5.9	-0.7	4.8
fdivp	-1.3	-2.2	-0.3	3.8	2.8	-0.8	1.3
fild	-4.3	-6.5	-6.1	-1.5	-0.8	-2.6	2.1
fstcw	-0.7	-1.2	-1.0	3.3	2.2	-0.4	0.2
imul	-3.3	1.3	-5.9	4.4	-1.4	-1.7	0.9
int	45.0	26.2	28.7	-1.8	-1.0	2.4	-1.4
nop	-2.3	-3.6	-3.2	-5.0	-1.6	4.5	-2.3
pushf	-2.4	-3.7	-1.8	-3.9	-2.2	-0.7	-2.6
rdtsc	-0.7	-1.2	-1.1	1.1	-0.7	3.8	-0.9
sbb	-6.5	-2.0	3.4	-2.2	0.3	0.8	-2.0
setb	-0.5	4.7	0.6	4.6	7.9	-0.3	2.1
setle	-1.0	-1.6	-1.4	-1.6	1.3	-0.6	-1.2
shld	-1.0	0.6	0.6	-1.1	-0.9	1.0	0.2
std	4.8	1.4	0.8	0.3	2.4	-0.6	4.8

Opcode Frequency

Higher
High
Similar
Low
Lower

Tests suggests opcode frequency roughly

1/10 lower
1/5 higher
7/10 same

vs goodware

Table 3: z-scores for frequent (top) and rare (bottom) opcodes

4. Extracting Opcodes:

The first step consisted of gathering random samples of malicious and non-malicious (‘goodware’) binaries. For *goodware*, sampling followed a two step process: An inventory of all PE exe files on a MS XP Home box was gathered by *Advanced Disk Catalog* (Elcomsoft, 2004). A preliminary binary file size distribution investigation yielded a log-normal distribution; for an in-depth explanation of the underlying generative processes, see (Mitzenmacher, 2003) and (Limpert, 2001). Twenty executables were uniformly sampled into four size blocks, five samples per block.
The size intervals were chosen as [0-10KB), [10-100K), [100-1K) and [1M-10M]; with square bracket and parenthesis denoting closed and open endpoints, respectively.
For *malware*, seven classes of interest were fixed (kernel-mode rootkit, user-mode rootkit, tool, bot, trojan, virus, and worm). Chris Ries’ collection (Ries, 2005) of 77 malware specimens was inventoried and 67 PE binaries (`exe` and `dll`) sampled into the seven classes of interest, with at least five unpacked samples per class. The malware specimens included variants of Apost, Banker, Nibu, Tarno, Beagle, Blaster, Frethem, Gibe, Inor, Klez, Mitgleider, MyDoom, MyLife, Netsky, Sasser, SDBot, Moega, Randex, Spybot, Pestlogger and Welchia.
Figure 1 illustrates the analysis workflow after the sample selection for malware; for goodware it follows essentially the same steps.
The samples were subsequently loaded into the de-facto industry standard disassembler, *IDA Pro* (DataRescue, 2006), in which a modified plugin, *InstructionCounter* (Porst, 2005), was run which extracted opcode statistics from the samples. An underground tool, *PEiD* (jibz, 2006), was used to augment the dataset with compiler and packer information, if applicable and identifiable. For goodware, a ‘functionality class’ (e.g. file utility, IDE, network utility, etc) was added manually to the dataset.
These datasets were parsed with a Java program and the *JA*va *MA*trix numerical analysis package (Mathworks, 2005). From the datasets, a list of opcodes was constructed, and the datasets normalized for further analysis in MS Excel. All this was run on MS Windows XP Pro in the virtual environment provided by *VMPlayer* (VMWare, 2005), following best practices in malware analysis (Szor, 2005, pp. 611-655).

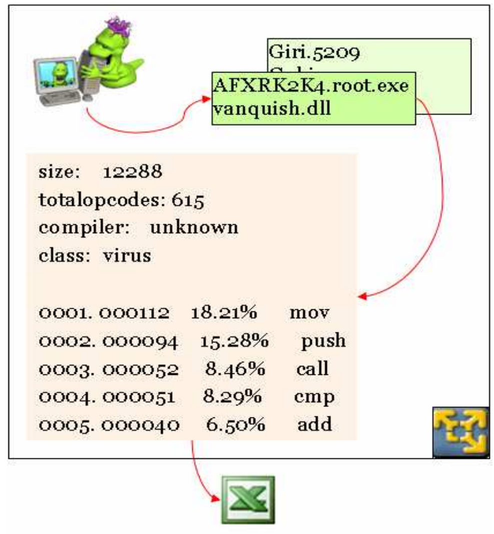

Figure 1: Analysis setup and workflow

4.1. Opcode Breakdown:

A list of opcodes was gathered from the samples and augmented with the entries in (Wikipedia, 2006), totalling 398 IA-32 opcodes.
The goodware samples yielded roughly 1.5m opcodes, and 192 different opcodes were found. 72 opcodes accounted for >99.8% of opcodes found, 14 opcodes accounted for ~90%, and the top 5 opcodes accounted for ~64% of extracted opcodes. Figure 2 shows the opcode breakdown graphically for the 14 most frequent opcodes for goodware.
The aggregate malware samples yielded roughly 665,000 opcodes. 141 different opcodes were found, including two undocumented ones, `salc` and `icebp`. Sixty opcodes accounted for >99.8% of opcodes found, 14 opcodes accounted for 92%, and the top 5 opcodes accounted for 65% of the extracted opcodes. Figure 3 shows the opcode breakdown for the 14 most frequent opcodes for malware (aggregated across classes).

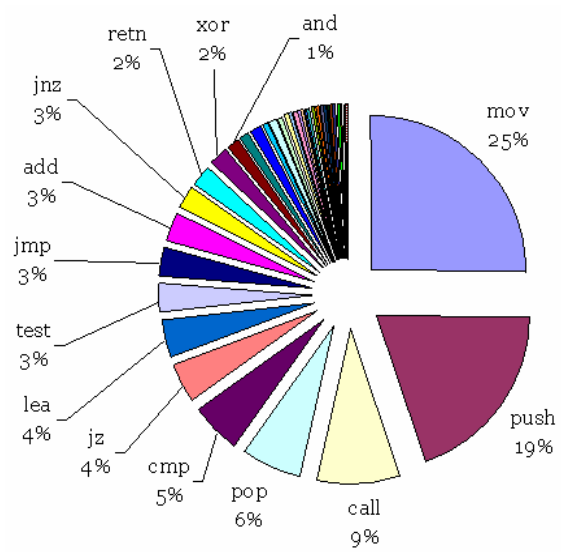

Figure 2: Most frequent 14 opcodes for goodware

We see that the top five listings for both malware and goodware are identical (`mov`, `push`, `call`, `pop`, `cmp`) and some minor rank permutations in the lower rankings.

A more granular proportional breakdown of the most frequent opcodes – specifically along the seven malware classes of interest - is shown in Table 1.

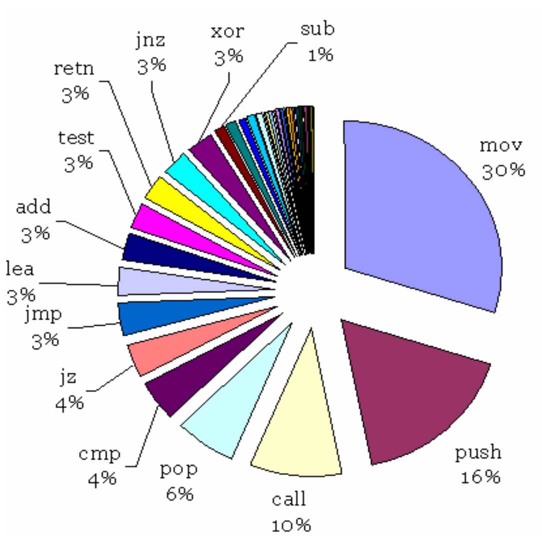

Figure 3: Most frequent 14 opcodes for malware

5. Statistical Analysis:

Frequency data in form of a 8*14 contingency table (rows: opcode, columns: binary classes) in and, three questions were formulated:

1a) Is there a statistically significant difference in opcode frequency between goodware and the seven malware classes of interest ?

1b) If there is a statistically significant difference, which opcode(s) is or are responsible for it?

2) How strong is the association between malware class and opcodes?

Statistical testing was used to shed some light on questions 1) and 2). Question 1a) was tested using Pearson's Chi Square procedure; for 1b) this was followed by a post-hoc standardized residual (STAR) testing of individual cells (Haberman, 1973).

The chi-square test examined the association between the row and column variables in a two-way table. The null hypothesis H_0 assumed no association between the variables (in other words, software class had no bearing on opcode frequency). The alternative hypothesis H_A claimed that some association existed.

The STAR post-hoc statistic is a z-score, asymptotically normal N(0,1) under the null hypothesis H_0 of independence, and indicates the H_0 fit for the individual cell. (Kim, 2006).

Question 2) was tested using Cramer's V, a measure of association strength or dependency between two categorical variables in a contingency table (Woo, 2005).

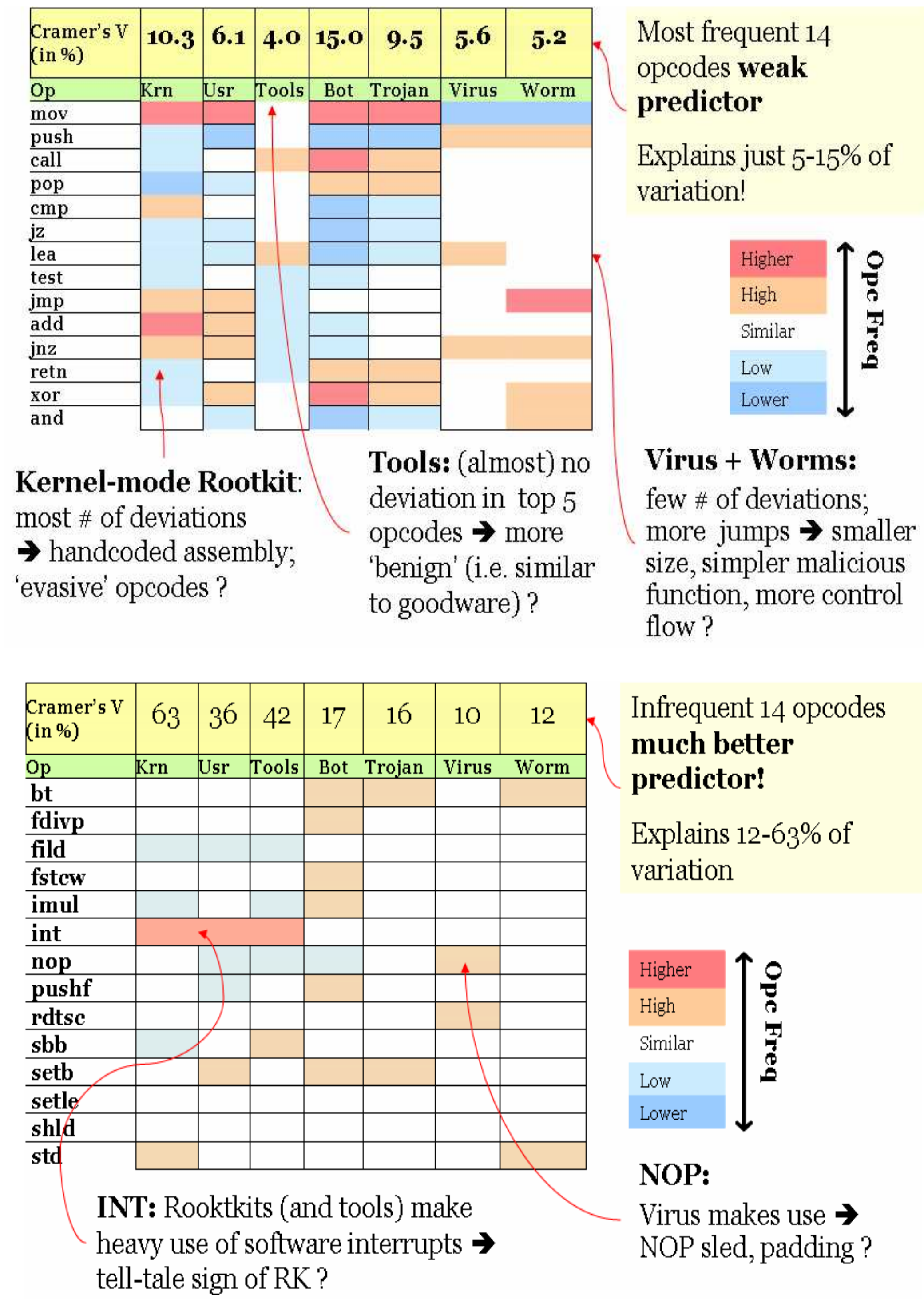

Table 4: Association strength between opcodes and malware classes. Rare opcodes (bottom) showed stronger association than more frequent ones (top).

5.1. Most Frequent Opcodes:

The first investigation focussed on the most frequent opcodes, as listed in Table 1. Table 3 (top) lists z-scores assessing opcode and malware class independence and Table 4 (top) shows the strength of the association. Cells are colour-coded for easier interpretation. The cut-off point for deviation was chosen as $z_c = 5$. White cells indicate that there is no significant deviation from H_0, bright red and blue indicate a much higher or lower occurrence of the particular opcode, as indicated by their very high or low z-scores.

Compared to non-malicious binaries, roughly 1/3 of the cells exhibited similar, 1/3 higher and 1/3 lower opcode frequencies. Speculations about these results were given in the side notes of Table 4 (top). It should be noted that these merit further investigation and should be taken as hypotheses.

5.2. Rare Opcodes:

Rare opcodes were not pruned akin to the most common opcodes; the frequency of the 14 rarest opcodes is zero for practically all cells. The rare opcodes listed in Table 2 were chosen uniformly at random among the population of opcode with frequency occurrences under 0.2% of total opcodes.

Table 3 (bottom) lists z-scores assessing opcode and malware class independence, and Table 4 (bottom) shows the strength of the association. Again, cells are colour-coded for easier interpretation. The cut-off point for deviation was chosen to be $z_c = 3$, more sensitive than for frequent opcodes because of the very small number of occurrences.

Compared to non-malicious execuatbles, roughly 70% of the cells exhibited similar, 30% higher and 10% lower opcode frequencies. Again, some preliminary hypotheses about the nature of these results were given in the side notes.

6. Discussion of Results:

Cramer's V can be interpreted as how much of the association can be explained without reference to other factors (Connor-Linton, 2003). For the case of the most common 14 opcodes, we see that opcodes were a relative weak predictor, explaining just 5-15% of the frequency variation. For the rarer 14 opcodes, the association was much stronger. The association between rare opcodes and malware explained 12-63% of the frequency variation (see Table 4).

In sum, malware opcode frequency distribution seems to deviate significantly from non-malicious software. Rarer opcodes seem also to explain more frequency variation then common ones.

7. Further Improvements:

Improvements to this approach can be undertaken along several lines. From the statistical testing point of view, further control procedures refinements for false discovery rate and type I errors, along the lines of (Kim, 2006, pp. 74-79) seem promising. Furthermore, the scope of the study could be broadened by analyzing *n*-way association (as opposed to 2-way) of factors.

Other factors beyond atomic opcodes such as compiler type (MS, Watcom, Delphi, gcc etc), opcodes classes (transfer, control flow, arithmetic, extensions etc) may yield some insight, as well. Inspired by the opcode-sequence based detection signatures of (Polychronakis, 2006), enriching the opcode factor beyond isolated opcodes to semantic 'nuggets' (positioned size-wise between atomic opcodes and basic blocks) may be a good idea.

Also, specific investigation of malware which implements conventional (Christodorescu, 2003) and 'targeted' obfuscation techniques (Yamauchi, 2006) may shed further light on the predictive value of opcode frequency distribution analysis. Finally, a time-series analysis of selected opcodes (like `nop`, `sysenter, icebp`) may be another way of discerning tell-tale trends and worth a try.

8. Malware on the Horizon:

It is hard to gauge how much mileage pattern-matching based AV detection techniques still have in them in light of these polymorphic and metamorphic threats. Some industry researchers are optimistic, maybe unduly so (Emm, 2007).
We briefly mention *k-ary malware*, a most worrisome development, in this context. K-ary malware, of which at this time only laboratory or very trivial examples are known to exist, seem able to elude conventional deployed defences *in principle*, not just in practice (Filiol, 2007).
This feat is accomplished by partitioning the malware's functionality spatio-temporally into *k* distinct parts, with each part containing merely an innocuous subset of the total instructions. In serial or parallel combination, they subsequently become active. Current AV models seem *unable to detect this threat* (or disinfect completely upon detection), which may be due to fundamental theoretical model assumptions (Filiol, 2006).
In light of existing and emerging malware, developing new models and methods is prudent. In the theoretical realm, this may entail moving beyond Turing machine models premised on the strong Church-Turing thesis ("computation-as-functions") towards more expressive models premised on "Interactive Computations" (Goldin, 2005). Interestingly, the necessity for a theoretical evolution was foreshadowed by Turing in his 1936 paper with his choice 'c-machine', as opposed to the standard automatic 'a-machine' (Turing, 1936).

9. Contributions of this Research:

We investigated opcode frequency distributions as a means to identify and differentiate malware. The scientific contribution of this research includes descriptive opcode frequency data for a medium-sized sample of malicious and non-malicious executables. The testing procedures went beyond standard Chi-Square tests in an attempt to isolate the opcodes that are most strongly associated with certain malware classes. Furthermore, we gave a quantitative statistical measure of how strong this association might be. The applications of these findings are of interest to several problem domains: AV scanners and intrusion prevention systems may get a fast first-pass criterion for on-demand, run-time execution and in-transit scanning.
Finally, these results and the synopsis of related work may stimulate further development and refinement of forensic tools such as Encase Forensics Law Enforcement (Guidance, 2006) and FTK (AccessData, 2005) for the benefit of law enforcement investigations and cyber-crime thwarting efforts.

10. Acknowledgments:

This paper is a revised and expanded version of a paper given at the 3rd International Conference in Global E-Security (London, UK). I thank Franklyn Turbak (Wellesley College) for germinating the time series idea and the anonymous reviewers for their helpful comments.

11. References:

AccessData Inc. *Forensic Toolkit*, Lindon, (UT), 2005

Bayer U., Kruegel C., Kirda E., "TTAnalyze: A Tool for Analyzing Malware", *Proceedings of the 15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR)*, Hamburg (Germany), April 2006

Bilar D., "Callgraph properties of executables", *AI Communications: Special Issue on Network Analysis in Natural Sciences and Engineering,* Amsterdam (NL), 2007

Christodorescu M., Jha S., "Static analysis of executables to detect malicious patterns", *Proceedings of the 12th USENIX Security Symposium*, Washington (DC), August 2003, pp. 169–186

Chinchani R., Berg E. V. D., "A fast static analysis approach to detect exploit code inside network flows", *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, Seattle (WA), September 2005

Clementi A., *Anti-Virus Comparative No. 13*, Innsbruck (Germany), February 2007, p. 7

Computer Economics Inc., *2007 Malware Report: The Economic Impact of Viruses, Spyware, Adware, Botnets, and Other Malicious Code*, Irvine (CA), 2007

Commtouch Inc., *Malware Report: Server-side Polymorphic Viruses Surge Past AV Defenses*, Netanya (Israel), 2007

Connor-Linton J., *Chi Square Tutorial*, Georgetown University, Washington (DC) March 2003

DataRescue sa/nv, *IDA - The Interactive Disassembler*, Liege (Belgium), v5.0.0.879, 2006

Elcomsoft Co. Ltd., *Advance Disk Catalog*, Moscow (Russia), v.1.51, October 2004

Emm D., "AV is alive and well", *Virus Bulletin*, Abingdon (UK), September 2007, p.2

Everitt B.S, *The Analysis of Contingency Tables*, Chapman & Hall, New York, February 1992 (2nd Ed.)

Filiol E, Helenius M., Zanero S, "Open problems in computer virology", *Journal in Computer Virology* (Vol 1. No.3&4), Paris (France) , March 2006, pp.55-66

Filiol, E, "Formalization and implementation aspects of K-ary (malicious) codes", *Journal in Computer Virology* (Vol. 3, No.2), Paris (France), June 2007

Flake H., "Compare, Port, Navigate", *Black Hat Europe 2005 Briefings and Training*, Amsterdam (Holland), March 2005

Goldin D., Wegner, P., "The Church-Turing Thesis: Breaking the myth", *Lecture Notes in Computer Science* (Vol. 3526), Berlin (Germany), 2005, pp. 152-168

Guidance Software, *Encase Forensics LE*, v. 5, Pasadena (CA), February 2006

Haberman S., "The Analysis of Residuals in Cross-Classified Tables", *Biometrics*, Vol. 29, No. 1, March 1973, pp. 205-220

Imbernon R., *Index your Files – Revolution!*, Malaga (Spain), v 2.6, June 2006

Jibz, Qwerton, snaker, xineohP, *PeID*, v0.94, May 2006

Kim S., Tsui K., Borodovsky M., "Multiple Hypothesis Testing in Large-Scale Contingency Tables: Inferring Pair-Wise Amino Acid Patterns in b-sheets", *Journal of Bioinformatics Research and Applications* (Vol 2, No. 2), 2006, pp.193-217

Kraus, J., *Selbstreproduktion bei Programmen*, Universität Dortmund Fachschaft Informatik, Diplomarbeit (unpublished), Dortmund (Germany), February 1980, pp.72-94

Kruegel C., Kirda E., Mutz D., Robertson W., Vigna G., "Polymorphic Worm Detection Using Structural Information of Executables", *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, Seattle (WA), September 2005

Li W., Wang K., Stolfo S., Herzog B., "Fileprints: Identifying File Types by n-gram Analysis", *Proceedings of the 2005 IEEE Workshop on Information Assurance*, USMA West Point (NY), June 2005

Limpert E., Stahel W., Abbt M., “Log-normal Distributions across the Sciences: Keys and Clues”, *BioScience* (Vol. 51, No. 5), May 2001, pp. 341–352

Mathworks Inc. and US National Institute of Standards, *JAMA : A Java Matrix Package*, v. 1.0.2, Natick (MA) and Gaithersburg (MD), July 2005

Mitzenmacher M., “Dynamic Models for File Sizes and Double Pareto Distributions”, *Internet Math* (Vol. 1, No. 3), 2003, pp. 305–333

Polychronakis M., Anagnostakis K., Markatos E.P., “Network-level Polymorphic Shellcode Detection using Emulation”, *Proceedings of the Third Conference on Detection of Intrusions and Malware & Vulnerability Assessment,* Berlin (Germany), July 2006

Ries C., *Automated identification of malicious code variants*, BA CS Honors Thesis (unpublished) , Colby College (ME), May 2005

Rozinov K., *Efficient Static Analysis of Executables for Detecting Malicious Behaviour*, M.Sc. Thesis (unpublished), Polytechnic University (NY), May 2005

Porst S., *InstructionCounter* (IDA plugin), Trier (Germany), November 2005

Spinellis D., “Reliable identification of bounded-length viruses is NP complete.”, *IEEE Transactions on Information Theory* (Vol 49, No.1), 2003, pp.280-284

Szor P., *The Art of Computer Virus Research and Defense*, Addison-Wesley Professional, Upper Saddle River (NJ), February 2005

Szor P., Ferrie P., “Hunting for Metamorphic”, *Proceedings of the 11th Virus Bulletin Conference*, Prague (Czech Republic), September 2001, pp. 123-144

Turing, A., “On computable numbers with an application to the Entscheidungsproblem “, *Proceedings of the London Math Society* (Vol 42, No. 2), 1936, pp. 230–265.

VMWare Inc., *VMWare Player*, v 1.01, Palo Alto (CA), December 2005

Weber M., Schmid M., Schatz M., Geyer D., “PEAT- A Toolkit for Detecting and Analyzing Malicious Software”, *Proceedings of the 18th Annual Computer Security Applications Conference*, Washington (DC), December 2002

Wikipedia, *X86_instruction_listings*, accessed June 6th, 2006

Woo C., “Cramer’s V”, *PlanetMath.org*, San Francisco (CA), April 2005

Yamauchi H., Kanzaki Y., Monden A., Nakamura M., and Matsumoto K., “Software obfuscation from crackers' viewpoint”, *Proceedings of the 2nd IASTED International Conference on Advances in Computer Science and Technology*, Puerto Vallarta (Mexico), January 2006, pp. 286-291

Zakon R., *Hobbes’ Internet Time Line*, v.8.2, North Conway (NH), November 2006