Enhancing software protection with poly-metamorphic code

Stephen Yip and Qing Zhao, University of Northumbria at Newcastle, England, UK

Stephen Yip is a senior lecturer in the School of Informatics, University of Northumbria at Newcastle, England, UK. He holds a PhD in Computer Science from the University of Durham and has research interests in software engineering, computer networks, web applications and security. Qing Zhou is a Master of Science student who has just completed his MSc dissertation on software protection and poly-metamorphism under the supervision of Dr Stephen Yip.

Abstract

We are arguing that legal protection alone is insufficient to uphold software copyright against infringements. Technical protection schemes are useful to safeguard software products against pirates and copyright violation. This paper focuses on software-based protection mechanisms, examining the commonly used "registration key" protection mechanism and its weaknesses in some details. It then describes our experiments with a new approach called poly-metamorphism – an approach to ensure that the appearance of the software protection code is altered each time the protected software is invoked or executed, making it very difficult for crackers to understand or modify the protection code.

The results and findings of our experiments are encouraging. For completeness, this paper also provides a brief review of hardware-based protection measures. Following the findings and evaluations of our work, we conclude that the poly-metamorphism approach is probably stronger than most of the existing software-based protection mechanisms. We intend to furnish our poly-metamorphic engine in the form of an Application Program Interface (**API**), and to make it available to researchers and software producers.

1 Introduction

Software piracy is the illegal use, duplication or distribution of a software product without the permission of its owner, violating copyrights or intellectual property rights (**IPRs**). "Global dollar losses due to software piracy increased 19% in 2002 to $13.08 billion, reflecting larger losses in a depressed software market" [BSA 2003]. Software piracy is seen to bear a significant impact on the entire IT industry [Aladdin 2003], despite the existence of laws on software copyright protection. Current laws protecting software copyrights are based on the agreement on Trade-Related Aspects of Intellectual Property Rights (**TRIPs agreement**) as part of the World Trade Organization (**WTO**) agreements. The TRIPs agreement [WTO 1994] clearly states, amongst its other provisions [Liu 2003], that:

(a) "Computer programs whether in source or object code" shall be protected as "literacy works" under the Berne Convention [Bennett 2000];

(b) Infringing copies of a copyright work must be liable to seizure by a member country of the WTO [Gikkas 1996];

(c) WTO members must provide effective action against infringement of IPRs protected in the TRIPs agreement, including remedies to prevent and deter further infringement [ADFAT 2003]; and

(d) WTO members are required to provide judicial authorities with power to issue injunctions, award damages, and to dispose goods tainted by infringement of the TRIPs agreement [WHOPL 1999], [Hunt 2003].

By examining these provisions, it is evident that the TRIPs agreement has provided a firm legal basis for the protection of software copyrights within the software industry. However, the legal protection of software is probably more effective with corporate users, than with individual home users whom the law is more difficult to police. In view of the proliferation of illegal copies of software available on the internet and CDs, it appears that legal protection alone might not be sufficient. "The legal rights to software protection does not provide complete power or control" [Chen 2001]. The argument is well known. No one has a right to enter your house without your consent. "The inviolability of your house is protected by law. Nevertheless, you prefer to have a lock in your door" [Ramanchauskas 1997]. Although software products are adequately protected by law, it is prudent to lock or protect software against piracy as computer software can be copied and distributed easily.

For many years now, the software industry has been striving to find ways to protect their software products. It has been argued that "technical protection schemes can diminish the number of illegal copies" [Hachez 1999]. Current technical protection schemes are either software-based or hardware-based.

2 Review of software-based protection measures

In the early days, licensed software was sold in the form of its container media - a set of floppy disks or CD. A piece of licence paper or a receipt provided the proof that the purchaser was authorised to run the software. In the case where the purchaser owned more than one computer, there was virtually nothing to stop a "single-use licence" owner installing more than one copy of the software on the owner's computers. It was also possible for a friend of a "single-use licence" owner to borrow a set of CDs or disks and install the software on the friend's computer. This scenario was quite common in the 1980s.

In the early 1990s, with the low production costs of CDs and the

availability of CD writers, software pirates began to manufacture illegal copies of licensed software. The authors of this paper witnessed, in the mid-1990s, CDs packed with pirated software being sold for as little as US$0.15 per CD in certain places in the Far East. During the 1990s, according to Liu [Liu 2003], IPR infringement was so widespread in China that the assumed piracy rate was 93%. The total losses in relation to the infringement of United States owner IPRs were estimated at US$1 billion in 1995 and US$2 billion in 1998.

Being aware of this problem, software vendors began to find ways to protect their software products. For example, Microsoft included a "Certificate of Authenticity" with their software distribution CDs bearing a so called product ID (e.g. 29395-OEM-0005952-04835). The idea was that an illegal copy of the software distribution CD could not be installed without the product ID being keyed in during the software installation process. Some other software packages required a serial number, an alpha-numeric key or a password to be entered before the software could be properly installed or run. These serial numbers, keys or passwords were often included within the delivery of the software package.

These were all primitive attempts towards software protection and in many cases quite a wide range of serial numbers or keys were accepted. Software pirates could get hold of a valid serial number or key by purchasing one legal copy of the licensed software, or by obtaining the information from a legal purchaser. A valid range of serial numbers or keys were then sold together with the pirated copies of the software. In order for an alpha-numeric key to be an effective protection mechanism, the key value must be specially generated (or tailor-made) according to the environment of each individual installation. Borrowing a valid key from one installation should not enable the software to be installed or used anywhere else.

2.1 The current approach of registration as a means of protection

The situation worsened with the advent of the internet, as software pirates began to put pirated software on websites for free download [Chen 2001]. To protect the copyright of their software products, software vendors came up with an idea called "registration", which also takes advantage of the internet.

Purchasers of licensed software are required to register the copy of software that is being installed in the purchaser's computer. The registration process is largely automated. The purchaser chooses to register by clicking a link to the software vendor's website. A registration program is then run to register the personal details (e.g. name, post code) of the purchaser, together with some details of the computer system (e.g. CPU / hard-disk serial number, etc) on which the software is being installed. Then a key or password is given by the registration program to enable the installed software to function. In these days of e-business, the purchaser is often required to enter a credit card number and to authorise payment for the software, before the registration key will be given to enable the software to function. As the registration key is generated according to the personal details and hardware characteristics of the installation, a copy of the registered software ported onto another computer would not work in its new environment, as the personal details and hardware characteristics would be different.

For example, popular "CD writer" software (NBR 5.5) can be freely down-loaded from the internet, even from its official website. It is a common phenomenon that many software vendors allow "free download" of their products from their websites. One has to emphasise that usually only the "download" is "free" – payment may be required (as part of the registration process) after the download and before the software can be installed or run on a computer. The reason for allowing free download is to attract potential customers and bring them a step closer to buying the software. This is particularly effective for small software vendors because it avoids the higher costs of advertising and distributing their products. In many cases, the downloaded software is an evaluation version that could only be used for a short period of time (e.g. 10 days) or only be executed or run for a fixed number of times (e.g. 5 times). Afterwards the evaluation version will display a message of "evaluation expired – please register and purchase the software".

Recent developments in registration number protection mechanisms have been successful in issuing registration keys that can only be used for one legal installation. Each registration key is tied down to both the personal details of the purchaser and the specific details of the computer system installed with a legal copy of the protected software. A valid registration key for one installation cannot be used anywhere else. However whenever there is a new protection scheme, there is always someone who can work out how to crack it. Then a new and stronger scheme will appear [Alfred 2002]. It is now possible for technically competent pirates to crack the registration protection mechanism.

2.2 Weaknesses of current registration protection mechanisms

It is observed that "many software producers now protect their programs by issuing registration numbers with each package. When you install the software, you must enter the registration number. However, this does not prevent all piracy, but limits it" [Jupitermedia 2003]. The fact is, it has not taken pirates a very long time to find ways to crack the registration protection scheme. There are two common approaches to violate registration protection:

(a) By analysing the object code of the protected software, and by observing the keys or passwords generated for a number of installations with known personal details and hardware characteristics, it is possible for pirates to write programs to generate valid keys or passwords for unregistered or illegal installation of the licensed

software. Having worked out how to generate valid registration keys for some popular software packages, some pirates put these so called "key gen" programs on the internet. It has been acknowledged that "the registration number protection technique presently has a serious problem: the publishing of registration keys on the internet" [Cerven 2002]; and

(b) Some pirates have the technical capability to crack software protection by modifying the program code. Their approach is often to make a minimal change (called a patch) to the executable code (also called binary or object code, i.e. not the source code which the pirates normally do not have). The patch is usually a very small change to bypass the code that checks the validity of the registration number entered.

As mentioned above, some software vendors provide free evaluation version of software packages which only work for a certain period of time or number of executions. To overcome the protection of these evaluation versions, there is a third, but legal approach. By removing and re-installing these evaluation versions, users may extend the trial period of the software without paying or registering. Although it seems to be a tedious task to repeatedly re-install the same piece of software, the actual process of removal and re-installation would usually take no more than a couple of minutes.

3 Crackers or reverse-engineers

Crackers are people who are interested in breaking software protection and security systems. Many crackers begin quite innocently because cracking is fun and challenging. There are others who attempt cracking as they want to use proprietary software packages without paying for them. Some are even involved in the illegal distribution of cracked software for financial gain. In order to produce better protection systems, it is important to understand commonly used cracking techniques.

Usually crackers are experienced programmers who can perform programming in low level assembly languages. Cracking requires technical knowledge in programming, cryptography, operating systems and reverse engineering.

"Reverse engineering" is working on existing lower level code, in order to generate higher level representation (i.e. higher level code information or knowledge) of the software being examined. The source code of protected software is normally unavailable to crackers, who have to work on the object or binary code to develop an understanding of the protected software.

3.1 Basic cracking tools

It is acknowledged that "virtually every serious cracker uses a debugger in order to break applications" [CrackZ 2000]. A debugger, as its name suggests, is a tool for programmers to debug their programs. A debugger allows a programmer to execute an object program one step (i.e. one machine instruction) at a time. Thus the programmer can observe the effect of each instruction, by examining any changes or effects on memory content or execution sequence as the result of each instruction being executed. Running a program under a debugger is like a slow motion playback, enabling a programmer to uncover/locate (i.e. debug) any programming errors within the code. A cracker can gain valuable insight about how the registration protection code works. Crackers can use a debugger to monitor contents of the CPU registers and memory locations while executing the object code. Debuggers allow crackers to trace or step through each line of code and modify values in memory locations. Crackers can also speed up their step-by-step observation of the program execution by allowing the program to run as normal, except stopping at a number of so called "breakpoints" selected by the cracker. "Dynamic analysis" is the name of this process of executing an object or binary code, in order to observe and understand the internal working of a program. Commonly used debuggers include Softice, Ollydbg, Windebug and Trw2000.

Another software tool used by crackers is called a "de-assembler". A de-assembler can translate machine code (in binary form) into a more readable or meaningful form called "assembler code" (which presents each machine code in mnemonics that are more meaningful to humans than the binary machine code). Assembler codes are not as readable as the source

Instruction	Explanation
1. call check_reg_key	: call routine to check registration key
2. cmp result, 1	: compare result returned by registration checking routine with the value of 1 (1 means registration key is valid)
3. jz reg_key_ok	: if result equals 1 (jz=jump if equal), jump to routine at address reg_key_ok
4. jmp reg_key_not_ok	: else jump to address reg_key_not_ok to print error message – "Registration key invalid"

Figure 1 – a section of assembly code checking the validity of the registration key entered

code of high level programming languages. However, software pirates are usually technical programmers who are familiar with programming low level programming languages called "assembly languages". "Static analysis" is the process of gaining an understanding of the internal workings of a program without executing the code, such as by translating machine code back to the more meaningful assembler code. Examples of de-assemblers commonly used by crackers include Win32dasm and IDA Pro.

It is sometimes possible to translate the object code to a high level programming language (i.e. at a higher level and more readable than assembly languages), by using a tool called a "de-compiler". Some common examples of de-compilers include Jad (for Java), Dede (for Delphi) and Exdec (for Visual Basic).

3.2 Bypassing software protection code

Crackers can aim to change a section of protected code to gain illegal access. They usually trace (with a debugger) the execution of the protected program to understand the code and then locate the section of code responsible for validating registration numbers. Finally crackers modify or patch the executable file using a hexadecimal text editor. To violate the protection scheme, crackers usually make a minimal code change to bypass registration number validation or its effect.

The following is an example of a very small but vital part of the protected software, in assembly code, that checks the validity of the registration key entered. If the registration key is ok, the program continues to perform its normal functions by jumping to the address labelled reg_key_ok. Otherwise, it jumps to a section of code at location reg_key_not_ok, which will print the message "Registration key invalid" and stops program execution.

Crackers can modify the jump instruction to bypass the protection scheme. For example, change the instruction at line 3 from a conditional jump instruction (jz – jump if equal) to an unconditional jump instruction (jmp). Thus the error processing (from line 4 onwards) will never be executed.

In practice, crackers usually focus on locating the jump instructions that are responsible for validating registration keys. With typical PC hardware, the following jump instructions are commonly used :

- JZ/JE	(jump if equal)
- JNZ/JNE	(jump if not equal)
- JMP	(unconditional jump)

Figure 2 – commonly used jump instructions in PC assembly languages

To narrow down the scopes of their searches for jump instructions, crackers usually set breakpoints at locations within the protected program that appear to be responsible for validating registration keys, then step through the code slowly with a debugger to locate a suitable point to modify the code (when running a program under a debugger, a break point will cause the execution to halt, so that the cracker can examine the effect of the section of code around the breakpoint, then continue execution in a slower step-by-step manner.)

3.3 General cracking steps

To summarise, crackers generally perform the following activities:

(a) Static analysis – crackers can gain an understanding of the protection scheme through studying the internal structure of the program. De-assemblers or de-compilers are tools that can help by translating the object or binary code back to a more meaningful form such as assembler code to high level programming languages;

(b) Dynamic analysis – to enhance understanding of the protected software by running the code under a debugger;

(c) Locating – to locate specific code sections responsible for registration validation, by setting breakpoints and stepping through code sections slowly. The aim is to find a suitable point to modify the code to bypass software protection; and

(d) Modifying - the final step is to change hexadecimal machine codes in the executable file (EXE or DLL format file) to bypass the protection scheme.

3.4 Techniques to enhance registration protection

There are a number of existing techniques that can be used to enhance the registration key protection mechanism:

(a) Code encryption

Software protection can be enhanced by distributing object codes in encrypted form [Ooi 2002]. The use of strong encryption algorithms and a longer encryption key length would help to secure software against crackers [Sinclair 2002].

(b) Compression

By applying compression to object code before software distribution. De-compression usually takes place at run time Crackers will find it more difficult to understand the code structure during static analysis, as it is in the form of a compressed file on disk.

(c) Code jumbling or obscuration

Special efforts to make source codes obscure, complicated or unintelligible as a measure against reverse engineering.

(d) Anti-debugging coding

Add special routines to detect and block crackers from using debugging tools against the protected software.

4 Metamorphism and poly-metamorphism

It has been said that "the evolution of metamorphic viruses is one of the great challenges of this decade" [Szor 2001]. "Metamorphic" is from the word metamorphosis – the change of form or character, as in the well known example of the metamorphosis of the silkworm into a butterfly. A

metamorphic virus is able to change its form or character and is therefore difficult to recognize and deal with [Perriot 2003]. Metamorphism, as exemplified in the case of the silkworm, involves the change or transformation into a different form or character (or a small number thereof).

Polymorphism goes a step further. The Longmans English Dictionary defines "polymorphic" as the existence in various (or many) different forms. The polymorphic techniques in programming allow an original piece of code to be changed (or mutated) into many different forms, whilst keeping its basic functions. Effectively, when a polymorphic code is loaded into memory for execution, it begins execution by transforming itself into a form different from its static form as found in a disk file. It is possible that the polymorphic code is transformed into a different form every time it is loaded from disk. Since it is the polymorphic code that transforms itself, as in the metamorphosis of the silkworm, we use the term "poly-metamorphism" to describe this phenomenon of self-transforming software.

The common approaches to implement polymorphism are :

(a) Insertion of redundant code to alter the appearance of the original code;

(b) Mutation – change or mutate existing code without affecting its basic function; and

(c) Varying the locations or sequence of existing code without affecting its basic function.

Some of these poly-metamorphic techniques have been used in computer viruses.

5 Poly-metamorphism as a strong means of protection

Current metamorphic techniques can achieve a high degree of variability. Our approach is to implement a poly-metamorphic engine that would take a piece of program code as input and perform mutations upon the original code to produce a resultant code piece (i.e. a mutant) that has exactly the same functions as the original program, but appears differently in individual instructions and sequences of code. The poly-metamorphic engine ensures that the appearance of the software protection code is altered each time the protected software is invoked or executed. When a pirate attempts to follow the execution path of the protected software in order to locate and bypass the registration validation code, the path (or sequence of execution) appears to be different every time. This makes it almost impossible for a cracker to locate and modify a certain line of code as to bypass registration validation. Poly-metamorphism "takes advantages of machine code assembled at random to yield extra-ordinary security against all kinds of attacks" [CyprotectAG 2003].

In our approach to achieve poly-metamorphism, at least part of the software being protected must be programmed in assembly languages (instead of high level programming languages). It is necessary to use an assembly language to prepare, dynamically (i.e. at run time), a matrix containing entry points or addresses of various routines within the software being protected. The entry addresses of various routines (and their positions within the entry-points matrix) are varied every time when the software is invoked and loaded into memory. This is to make it difficult for crackers to locate the routines responsible for the validation of registration numbers.

The path (the correct execution sequence) of the "registration key validation" code is deliberately hidden (or made indirect) from the cracker. Functions within the "registration key validation" code are called indirectly via this matrix of entry points. Inside this matrix, entry points for other functions (i.e. for normal processing) and entry points of routines responsible for "registration key validation" and entry points of non-existing or fake functions / routines are all fixed together. To make it more

Test program	Challenge posted to the following "crackme" website	No. of viewers	No. of downloads	No of replies received
Crackme1	http://crack.zuasoft.com	N/A	N/A	23
Crackme1	http://www.crackmes.de	N/A	N/A	N/A
Crackme1	http://board.anticrack.de	379	51	16
Crackme1	http://picasso.poupe.net	49	10	0
Crackme1	http://www.chinaecg.com	35	N/A	4
Crackme1	http://www.chinaycg.com	9	N/A	0
Crackme1	http://www.exetools.com	52	12	4
Crackme1	http://www.crackbest.com	45	N/A	6
Crackme2	http://crack.zuasoft.com	N/A	N/A	3
Crackme2	http://board.anticrack.de	233	30	9
Crackme2	http://www.ahzol.com	211	N/A	13
Crackme2	http://www.crackmes.de	N/A	N/A	N/A
Crackme2	http://picasso.poupe.net	199	83	9
Crackme2	http://tntforum.com	21	N/A	0

Figure 3 – Statistics of two "crack-me" challenges posted on a number of websites

difficult to follow the execution path of the "registration key validation code", every time the protected software is loaded into memory, a different pattern of arrangement of entry points, within the matrix, is randomly generated. In order to protect the "registration key validation code" even further, the whole program (i.e. the protected software) is encrypted, making it difficult for crackers to locate vulnerable instructions (such as conditional jump instructions [je] or [jne]) via static analysis. Moreover, even the matrix of entry points itself is encrypted. Encryption has made static analysis almost impossible and poly-metamorphism has made dynamic or runtime analysis very confusing or difficult.

6 Experiments with poly-metamorphism for software protection

In order to evaluate poly-metamorphism as a means of software protection, we have implemented two versions of registration number protection systems. The first version is called Crackme1 (which only uses encryption without poly-metamorphism) The second version is called Crackme2 (which employs both poly-metamorphism and encryption). The two systems were posted as challenges in a number of "crack-me" websites for about a month from August to September 2003. The following table shows the statistics about the number of viewers of the challenges at different websites, together with number of times that the test programs were downloaded, and the number of replies (or messages) received from people who were interested after viewing the challenges.

The overall result is that only 3 people managed to crack Crackme1 (which only uses encryption without poly-metamorphism). Although quite a number of emails from interested crackers, no one managed to crack Crackme2 (which employs both poly-metamorphism and encryption). For further details of our experiment (including design and implementation of our Poly-metamorphic engine), please refer to [Zhou 2003].

7 Hardware-based protection measures

Some hardware-based protection mechanisms are reviewed below :

(a) Hardware-key – hardware key as a device is connected to an input / output port (serial, parallel or the new USB ports) on the computer. The protected program checks the I/O ports for a valid hardware key before it would continue normal execution.

(b) Smart cards – the protected program, as it begins execution, checks whether a valid smart card is present.

(c) CD check – applicable to software distributed on CDs. The program checks to identify if it is running off a valid CD. This CD protection mechanism has been used by some vendors of computer games.

8 Conclusions and future work

We have argued that legal protection alone is insufficient to uphold software copyright against infringements. After examining existing software protection mechanisms in general, and registration key protection in particular, we propose poly-metamorphism as a strong means to enhance software protection. The results and findings of our experiments are encouraging. Our poly-metamorphic engine is able to automatically generate mutants with code sequences different from the original program, but with the same functionalities. It would be difficult even for the author of the original software package to locate the registration validating code at run time, and more difficult or impossible to bypass or violate its protection mechanism. Perhaps "the poly-metamorphic method is among the strongest ciphers available today and it's probably the strongest." [CyprotectAG 2003]

"Metamorphism is a technology with a very promising future; ... none of the

Type	Advantages	Disadvantages
CD check	The software can only be installed or executed from the original cd-rom	User needs to keep the original cd-rom. Sometimes the specific media may be counterfeited or duplicated by software or hardware means.
Hardware key	The protected software can only be executed with the hardware key (dongle) plugged in the computer. It is difficult to duplicate.	Extra cost for users to possess the hardware key.
Smart cards	The protected software can only be executed with the smart card. With built-in microchips, smart card can execute checking routines, access or update internal data.	Extra cost of a smart card and a smart card reader.

Figure 4 – Hardware-based protection mechanisms

protections currently offered use full metamorphism." [Cerven 2002] Our future work is to furnish the poly-metamorphic engine in the form of an API, and to make it available to researchers and software producers.

References

[ADFAT 2003] Australian Department of Foreign Affairs & Trade (2003) *China's WTO membership - the accession negotiations* 19 March as found at http://www.dfat.gov.au/trade/negotiations/accession/wto_china.html

[Aladdin 2003] Technical white paper, *Software Protection–The Need, the Solutions, and the Rewards*, Aladdin Knowledge Systems. 2003. Available at: http://www.eAladdin.com/hasp

[Alfred 2002] Alfred K.M. Lo, *Software Protection and its Annihilation*, University of Birmingham May 2002

[Bennett 2000] Bennett, D., Liu, X., Parker, D., Steward, F., and Vaidya, K. (2000) *Technology transfer to China: a study of strategy in 20 EU industrial countries*, http://research.abs.aston.ac.uk/working_papers/0006.pdf

[BSA 2003] *Piracy Study* Business Software Alliance June 2003 Available at: http://global.bsa.org/usa/research/

[Cerven 2002] Pavol Cerven, *Crackproof Your Software—the Best Ways to ProtectYour Software against Crackers*, No Starch Press 2002

[Chen 2001] Min Chen, *Software Product Protection*, Seminar on Network Security Helsinki University 2001, page 2

[CyProtectAG 2003] CyProtectAG, *The polymorphic encryption method*, http://english.cyprotect.com/main0110.php, 2003

[Esprit 2000] Esprit Project, *State of the Art in Software Protection*, Filigrane Consortium, May 2000

[Ooi 2002] K.S.Ooi, Brain Chin Vito, *Cryptanalysis of S-DES*, University of Sheffield Centre, Taylor's College, April 2002

[Gikkas 1996] Gikkas, N. S. (1996) *International licensing of intellectual property: the promise and the peril* Journal of Technology Law & Policy V1(1), http://journal.law.ufl.edu/~techlaw/1/gikkas.html

[Hachez 1999] G. Hachez and C.Vasserot , *State of the Art in Software Protection*, FILIGRANE Consortium, 1999, http://www.dice.ucl.ac.be/crypto

[Hunt 2002] Hunt & Hunt Lawyers, *Enter the dragon: China and the WTO* China Focus Issue, 2002, http://www.hunthunt.com.au/hunthunt/Publications/ChinaFocusFeb02.pdf

[Jupitermedia 2003] Jupitermedia, *Copy Protection*, 2003. http://www.jupitermedia.com/TERM/C/copy_protection.html

[Liu 2003] Vincent Liu, *Copyright and software protection: is it working in China?*, Computers & Law, Issue 51, March 2003

[Perriot 2003] Frederic Perriot, *Striking Similarities: Win32/Simile and Metamorphic Virus Code*, Symantec Corporation 2003

[Ramanchauskas 1997] Vitas Ramanchauskas, *Protect your programs from piracy*, LastBit Software Ltd 1997

[Sinclair 2002] James Sinclair, *Securing electronic data transfer – responsibly*, GNSS 2002, http://www.gnss.com

[Szor 2001] Peter Szor and Peter Ferrie. *Hunting For Metamorphic*, Virus Bulletin Conference, September 2001

[WHOPL 1999] White House Office of Public Liaison, *Briefing on the Clinton administration agenda for the World Trade Organization material - Summary of US - China bilateral WTO Agreement*, 17 November 1999, http://www.uschina.org/public/991115a.html

[WTO 1994] *WTO Agreement on Trade-Related Aspects of Intellectual Property Rights*, signed in Marrakesh, Morocco on 15 April 1994, http://www.wto.org/english/docs_e/legal_e/27-trips_01_e.htm

[Zhou 2003] Qing Zhou, *Improving Registration Number Protection Mechanism with the Poly-Metamorphism Algorithm*, MSc Dissertation, School of Informatics, University of Northumbria, Newcastle, England, UK, Sept 2003