Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the BRADLEY Virus[1]

Major Eric Filiol
Army Signals Academy – Virology and Cryptology Laboratory[2]

About Author(s)

Eric Filiol is Head Scientist Officer of the Virology and Cryptology Laboratory.
Contact Details: c/o Ecole Supérieure et d'Application des Transmissions, Laboratoire de virologie et de cryptologie , B.P. 18, 35998 Rennes, France, phone +33-2-99843609, fax +33-2-99843609, e-mail efiliol@esat.terre.defense.gouv.fr

Keywords

Encryption, viral encryption, antiviral techniques, code disassembly, key management.

Reference

Reference to this paper should be made as follows:

Filiol E. (May 2005). Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the Bradley Virus. In Turner, Paul & Broucek, Vlasti (eds.), EICAR Best Paper Proceedings, CD-ISBN87-987271-7-6, pp.216-227.

[1] This paper was presented at the EICAR2005 annual conference held in St.Juliens/Valletta (Malta), April30 May3, 2005 and published in the Best Paper Proceedings CD Turner, Paul & Broucek, Vlasti (eds.). ISBN87-987271-7-6

[2] This paper may be downloaded at: WebUrbBrief http://papers.weburb.org/frame.php?loc=archive/00000136/

Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the BRADLEY Virus

Abstract

Imagining what the nature of future viral attacks might look like is the key to successfully protecting against them. This paper discusses how cryptography and key management techniques may definitively checkmate antiviral analysis and mechanisms. We present a generic virus, denoted Bradley which protects its code with a very secure, high speed symmetric encryption. Since the main drawback of using encryption in that case lies on the existence of the secret key or information about it within the viral code, we show how to bypass this limitation by using suitable key management techniques. Finally, we show that the complexity of the Bradley code analysis is at least as high as that of the cryptanalysis of its underlying encryption algorithm.

Introduction

Antiviral detection is directly based on the capability to have malware codes at one's disposal and to study them by disassembly means. Thus, viral databases can be updated and antiviral engines can be upgraded.

A few malware writers try to make this task more difficult by implementing various techniques which aim at delaying the knowledge and the understanding of their codes: obfuscating, rewriting, encryption.... These codes are denoted *armoured* codes. The first and most famous one is probably the *Whale* virus appeared in the early nineties. More recently, the *MyDoom* virus very naively tries to complicate antiviral experts' work by implementing basic encryption techniques. Up to now, none of the known malware succeeded in preventing code analysis.

The main explanation for this failure lies on two facts:

- antiviral experts always manage to obtain a malware copy (infected file). As they are widely disseminated, malware code samples (viruses, worms...) are always very easily available. This comes from the fact that limited virulence[3] is not a feature inherent to malicious codes.
- When present, techniques aiming at making code analysis more difficult are bound to fail. The main reason is that the related problems (that is to say, problems to be solved in order to bypass code protection) belong to polynomial complexity class. As an example, encryption techniques are always relatively easy to break since the key space is too limited and allows an exhaustive search approach. Moreover, encryption algorithms that have been found in known malware codes are either very naive or do not offer high level of security.

In this paper, we present a new concept about malicious codes combining efficient key management with high-level security encryption algorithm. Different analysis and experiments have confirmed the impossibility to study the code, under the assumption that we managed to get a copy of it. By limiting the code presence and virulence in the computer, we show also how to make this assumption very unlikely. We illustrate these concepts (proof-of-concept) by presenting from an algorithmic point of view the most simple example of a new virus family called the *BRADLEY*

[3] *Virulence* is an index measuring the level of risk for self-reproducing codes. This index, hence the risk itself, is related to the number of copies of the malware code. For details, see (Filiol, 2003).

viruses[4]. As a main result, we show that the general problem of *BRADLEY* code analysis is equivalent to the cryptanalysis of a secure encryption algorithm in the sense that it is exponentially complex.

This paper is organized as follows. In Section 2, we first define precisely the background and the known cases where attempts to use cryptography in viral codes have been made. We show why all this attempts were bound to fail. Section 3 recalls key management techniques presented in (Riordan & Schneier,1998). In Section 4, we present the generic viral concept[5] using strong encryption combined with optimal key generation and key management. At last, Section 5 proves that *BRADLEY* viral code analysis is equivalent to encryption systems cryptanalysis and that it is, in fact, of exponential complexity. Conclusion will address the problem of fighting against such armoured malware.

The purpose of this paper is purely academic and draws our attention on the evolution of viral risks. It shows how the malware risk may evolve very quickly (if not already the case) and cause great concern among the antiviral community. This is the reason why we will not give any detailed code. The activity of our laboratory is dedicated to defensive aspects. Our mission consists in identifying new risks, in testing them in practice and assessing the level of potential threat precisely. All experiments are conducted in a P4-like laboratory, while remaining strictly within all existing laws.

Definition and Background

Computer virology

The reader is supposed to be familiar with basic definitions about malware (virus, worms, trojan horses...) and antiviral techniques. We just recall the following starting definition of *armoured* codes.

Definition *(Armoured codes)*

An armoured code is a program which contains instructions whose goal is to delay, complicate or forbid its own analysis during either its execution or through its disassembly.

The best known example is probably the *Whale* virus which appeared in september 1990. The virus did actually represent a very limited risk but it intended obviously to make its analysis very difficult. Its code contains roughly a dozen of program traps and tricks hampering trace, disassembling and code analysis: dynamic decryption/encryption, code obfuscation, code nesting...Once activated, the viral code tries to detect the potential use of a debugger and consequently freezes the keyboard. Using polymorphism techniques, about 30 different random variants were possible for an infected file.
What the *Whale* virus easily managed to cause is not a terrific epidemic but a waste of anti-virus experts' time and a nearly three-day delay to eradicate it. Nowadays, the main part of the viral action is completed during the first thirty minutes after the beginning of the infection (a good example could be the *Slammer* worm which appeared in january 2003); therefore, such a delay in code analysis cannot be acceptable. That is the reason why *armouring* code techniques must be seriously taken into account.

These techniques can be divided into different classes, as follows.

[4] This name refers to one of the most famous US (armoured) main battle tank.

[5] Without loss of generality, we will use the general term "*virus*" but everything presented in this paper may apply to any other malware type: trojans, worms, logical bombs...

- *Code obfuscation:* the aim is to transform a program into another which both is functionally identical to the original and more difficult to uncompile and reverse engineer. In other words, the program is written in order to reduce readability and understandability. Three types of transformations are generally used: lexical transformations (variable name exchange), control flow transformations (making the program flow more complex by using code nesting or placebo code) and data flow transformations (action on data structures by changing storage, encoding, aggregation and order of data). More details can be found in (Chow et al, 2001 ; Collberg & Thomborson, 2002 ; Flanders, 2003 ; Shah, 2002). In a viral context, these techniques are of limited interest. Obfuscated codes are generally too slow and of too large a size to be efficiently used by undetectable malware. Moreover, the results presented in (Barak et al, 2001) show there is no transformation able to prevent the code of every program from revealing any other information except the program's input-output behaviour.
- *Polymorphism*: the aim is to make the code change as often as possible by using rewriting techniques (equivalent functionality but different code). The code analyst has thus to face up not a single version of the malware code but several versions of it, produced by mutation-like operations; hence the dfficulty to efficiently fight against it. Fortunately, code analysis becomes always possible in the end and, consequently, all polymorphic techniques are overcome. A detailed discussion on polymorphism is presented in (Pearce, 2003).
- *Encryption*: the purpose is both to provide polymorphism (encrypted code change with every different key) and to prevent code analysis. So far, known techniques suffer from a lack of efficient key management. The key must be securely available to the malware code only. Hardware solutions which are generally envisaged to prevent code analysis are not suitable for mobile codes like virus or worms. Mostly, the key elements are somehow or other contained in the code itself.

As these first two techniques are concerned, code analysts fortunately will always get to the end of it since these techniques always produce deterministic results (even if some algorithms may be partly probabilistic). The only thing is that the analysts need time to study the code behaviour instruction by instruction. Therefore, this study is likely to be time-greedy and requires many human resources and much effort dedication. To sum up, none of these techniques are suitable to definitively protect a code against its analysis.
Encryption is maybe less obviously easy to handle contrary to what experience tells us so far. Fortunately, only naive or very unsecure encryption methods have been used in known malware (Ciubotariu, 2003): constant masking (like in earlier macro-viruses for example), rot13 encryption or other weak encryption systems (as an example, *DarkParanoid* virus used very simple arithmetic functions – ADD, SUB, XOR, NEG, NOT, ROL, ROR – as encryption functions).... Besides, weak key management always allows to recover the key and then to decipher the malware code when dealing with strong cryptosystems.

Even when combining these techniques (one the most famous example is probably the *Zmist* virus (Ferrie & Zsör, 2001), the virus analyst managed to derive all the virus action. All these techniques are definitively deterministic

Cryptology

The reader is supposed to be familiar with basic concepts of cryptology as well. A detailed monography about cryptography will be found in (Menezes et al, 1997). We will just recall previous uses of cryptology inside malware and a few useful concepts we will use throughout this paper.

Cryptology has previously been envisaged to provide computer virology with very efficient tools. On the one hand, cryptographic techniques have recently been considered as a means for optimal worm propagation (Balepin, 2003). The use of cryptographic hash functions, for instance, is suitable for speeding up *Curious Yellow worm* propagation (Wiley, 2002).
On the other hand, the combination of cryptographic techniques with viral technologies led in 1996 to the concept of "*Cryptovirology*" as presented in (Young & Yung, 1996, 2004). Cryptovirology consists in applying cryptography tools to malicious codes in order to strengthen, improve or develop such codes.
Particularly, cryptography appears to be very efficient in designing payloads. Several convincing examples are presented in (Young & Yung, 1996). The main goal is to make a victim host dependent upon the virus – *i.e.* a virus can survive in the host if it makes the host depend in a critical way on the very presence of the virus itself. These results are mainly obtained with public-key cryptographic techniques[6] combined with limited symmetric cryptography techniques.
Though very efficient, these approaches aim only at protecting the action of the virus (the *payload* result) but not the virus itself. In other words, if a copy of a cryptovirus is somehow or other obtained and analyzed by reverse engineering, none of the cryptographic tools it contains will totally protect them against its code analysis. Thus, the exact knowledge of the code is likely to allow antiviral software update and limit/forbid the malware's action. The main limitation comes from the fact that cryptovirus as defined by Young and Yung, is not able to manage secret key part in a suitable and efficient way for that particular purpose.

The basic technique we discuss in this paper can effectively forbid such code analysis and thus, properly complement all the approaches developed in (Young & Yung, 1996, 2004). Other more sophisticated techniques are being tested in our laboratory.

Environmental Key Generation

Malware are mobile agents by nature. If they pass through an "insecure network" or environment (from the malware's point of view), they may be analyzed (disassembled) so that their code will be completely accessible to the attacker (the analyst). As previously explained, traditional encryption systems are dealing with static keys. Actually, the key is present somehow or other in the agent (hardware or software).

In 1998, B. Schneier and J. Riordan (1998) introduced the notion of *environmental key generation* to address this problem. In other words, keying material is constructed from certain classes of environmental data. Environmental key generation can thus be useful when the sender wishes to communicate with the receiver such that the receiver could only receive the message if some environ-mental conditions are true. Environmental key generation can even be used in circumstances where the receiver is not aware of the specific environmental conditions that the sender wants his communication to depend on. This latter case corresponds exactly to our malware code analysis problem. The receiver here is the malware code present in a computer (the environment) and the sender is malware code author or the target system itself.
The difficulty with building an environmental key generation protocol is that the threat model assumes that any attacker (the malware code analyst) has total control over the environment. All information available to the malware program can be found by the attacker as well. All inputs to the program are supplied by the attacker and the program states themselves are completely determined

[6] A *cryptovirus* is defined as a computer virus that contains and uses a *public key*.

by the attacker during the code analysis. As such, the constructions must resist direct analysis and dictionary attacks in the form of Cartesian deception, that is to say, in which the attacker tells lies about the environment.
Riordan and Schneier (1998) discuss several constructions for environmental key generation. To illustrate their approach, let us consider the following basic construction. Let N be an integer corresponding to an environmental observation, H a one-way function (typically a hash function), M the hash of the observation N , $\oplus$ the bitwise exclusive-or operator, || the concatenation operator, R a nonce and K a key. The value M is carried by the agent (the malware code in our case). Hash function can be used to conduct tests and construct the keys so that examination of the agent does not reveal the required environmental information. Then possible constructions, among many others, are:

1• if **H(N) = M** then let $K = N$.
2• if $H(H(N)) = M$ then let $K = H(N)$.
3• if $H(N_i) = M_i$ then let $K = H(N_1,N_2,...,N_i)$.
4• if $H(N) = M$ then let $K = H(R_1,N) \oplus R2$.

Let us note that the first construction is used in most of static encrypted password authentication schemes. The most important feature of each of these constructions is that knowledge of M does not leak any information on K.
Riordan and Schneier proposed several efficient constructions which provide efficient environmental key generation protocols using various techniques: thresholding (protocol using the ideas of cryptographic secret sharing), nesting (action of the mobile agent is ruled by several environmental keys used in the sequential way), time indexation (part of the environmental date required to generate the key are based on time)...
Environmental key generation has only been proposed from a theoretical point of view by the authors. Some aspects still need to be thoroughly tested. Particularly, for most of the constructions they proposed, the attacker is likely to find the key by observing both the agent and the environment. The search space for the activation data may always be small enough to allow an exhaustive search approach. Moreover, by observing mobile agent actions, the attacker may easily determine where and which kind of data the agent is interested in. That implies that a patient analyst will obtain information about the agent at the same time this latter is activated by the suitable environmental data.
We now present a practical and efficient use of environmental key generation in the case of viral code armouring.

A Generic Armoured Virus: the BRADLEY virus

Let us discuss now the generic family virus named BRADLEY. Without loss of generality, we choose to describe only a basic but powerful example. Some more complex protocols have been developed or are currently under study (see Section 6). Two different codes have been developed and tested:

- a directed but generic virus which aims at specifically infecting a given group a machine/people (variant A),
- a directed virus dedicated to specifically strike only any given user (variant B).

Minor variants have been tested as well and will be listed later on. The codes have been designed both for Windows and Unix systems. They successfully managed to bypass antiviral software which

all remained silent. Since BRADLEY viruses are only proof-of-concept viruses, we will focus only on the armoring protocol part. Complete source code is not available. The general structure of the codes is given in Figure 1 and summarized as follows

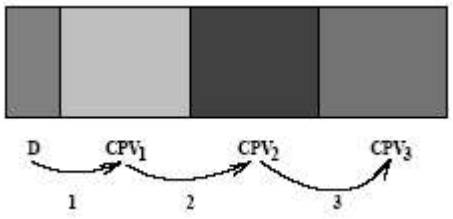

Figure 1 : Overall structure of BRADLEY codes

- a decipherment procedure ***D*** (the *decryptor* which purpose is to collect activation data, test and evaluate them and finally decipher the different encrypted parts of the code;

- a first encrypted part EVP_1 with encryption key K_1. Once deciphered (under the name CPV_1), this part installs all anti-antiviral functions (passive and active) ;

- a second encrypted part EVP_2 with encryption key K_2.This part (deciphered as CPV_2) contains the infection functions and the polymorphism procedures. When replicating, the virus will always and completely change its form (including the decipherment procedure) ;
- a third part EVP_3 (optional; deciphered as CPV_3) with encryption key K_3.It contains the payload functions (in our case, a simple opening window issuing a infection warning in order to keep control over the virus).

Note that these three encrypted parts are exactly of the same size in order to give the slightest information on the underlying code.

Let us now describe the key management protocol. The activation data – in other words the data required to construct the different keys – are (variant A):

- the local DNS address (e.g. *@company.com*), denoted α,
- current system time (hours hh only) and date (mm dd), denoted δ,
- a particular data present in the target system(s) (in our case a particular file), denoted ι,
- a particular information under viral code author control, located outside the system (public channel) but easily accessible to the virus (in our case, a given web page containing a particular value whose presence is limited in time and related to the value δ); it is denoted π and obtained from the hash of this information[7].

[7] One may object that the presence of the webpage url within the procedure ***D*** could give a useful information to the analyst. Since the webpage is under malware author's control (e.g. located in a rogue state), it is a very dubious hypothesis that the analyst could successfully access to the suitable activation data, especially if the data availability is very

For the variant B, data ι is a given public key which is present in a *pubring.gpg* for example. Thus, the virus may target a particular user or users communicating through encrypted emails/data with any given user. The viral code uses the hash function SHA-1 (NIST, 1993) as one-way function (here denoted H). Then, the environmental key protocol is described as follows:

1. the decipherment procedure D collects the activation data either directly (α, δ and π) or repeatedly[8](ι) and compute a 160-bit value V given by $H(H(\alpha \oplus \delta \oplus \iota \oplus \pi) \oplus v)$ where v is the first 512 bits of EVP_1 (in its encrypted form) ;
2. if $V = M$ where M is the *activation value* contained in the viral code, then $K_1 = H(\alpha \oplus \delta \oplus \iota \oplus \pi)$ otherwise the decipherment procedure stops and disinfects the present system from the whole viral code.
3. D deciphers EVP_1 producing $VP_1 = DK_1(EVP_1)$ and launches it. Then D is computing $K_2 = H(K_1 \oplus v_2)$ where v_2 is the 512 last bits of VP_1.
4. D deciphers EVP_2 producing $VP_2 = DK_2(EVP_2)$ and launches it. Then D is computing $K_3 = H(K_1 \oplus K_2 \oplus v_3)$ where v_3 is the 512 last bits of VP_2.
5. D deciphers EVP_3 producing $VP_3 = DK_3(EVP_3)$ and launches it.
6. After virus action is completed, the virus disinfects itself totally.

Some remarks can be made about this protocol:

- from replication to replication, the whole code (including procedure D and value M) has completely changed every time. This implies a total control of the polymorphic procedure relatively to the key management protocol by the author of the viral code (*i.e* .the evolution of the activation data – in practice only the values δ and/or π) ;
- the purpose of values v_i is to make the data span the whole input space (512 bits) ;
- the different parts VP_i may be compressed before encryption ;
- the keys K_1, K_2 and K_3 can be made independant by using additional environ-mental data;
- the autodisinfection may be delayed in order to handle the time and date values in a less strictly way. In that case, the decipherment procedure D remains active in system memory.

Other variants have been tested as well, particularly to produce the most optimal code in terms of size and stealthiness. The most significative variant includes the following features:

- the underlying code is compressed,
- instead of embedding compression and encryption functions within the virus code, this latter will borrow local resources if present,

limited in time. Nonetheless, we have developed a variant of the basic protocol which is discussed in this section. Instead of only one data π, we use two external activation data π and π'. Each of them come from two different webpages. The second webpage's url is encrypted by means of the key contructed from data α, δ, ι and π. Once decrypted, the virus gets the second activation data π' and a secret permutation function P (at the very beginning of EVP_1). Finally, the key K_1 is build from data $P(\alpha)$, $P(\delta)$, $P(\iota)$ and π' in the same way as key K_1 is. In this variant, $K1$ is superseded by K_1 .

[8] "*Repeatedly*'' means here that the virus scans any data contained in the system. In our case (a given file), the virus looks recursively for that data through the tree file system.

- that implies that one more activation data is required and repeatedly scanned for existence or not of compression and encryption softwares.

For all variants we developed, encryption algorithms that have been used are RC4 (Rivest, 1992) and RC6 (Rivest et al., 1998) while gzip compression has been chosen.

Viral Code Analysis and Cryptanalysis

To evaluate the code analysis complexity, two cases must be considered:

- the analyst has the viral binaries at one's disposal,
- the analyst does not have them.

The second case is more likely to happen if we consider that, in any case, the virus limits its presence inside the target system by disinfecting itself from it.

But let us suppose that the analyst, even if it is very unlikely, has managed to get one copy of the virus binaries. One could object that *rootkit* or *honeypots* may be used to trap a copy of the code and then analyze it. Thus the environmental key could be derived. In our model, this solution remains impossible in practice to use: we use viral codes fitted for dedicated attacks (wiretapping viruses, dedicated destruction of data...). To have a significant probability of catching at least a copy of such a virus, a huge number a machines should be equipped with either rootkits or honeypots. This approach is thus inapplicable. On the contrary, armoured, generic worms codes would be more easy to catch since the worm spread generates millions of viral code copies.

So let us show that the environmental key generation protocol presented in section 4 efficiently forbids code analysis unless a cryptanalysis problem of exponential complexity is solved.

Proposition

The analysis of a code protected by the environmental key generation protocol defined in Section 4 is a problem which has exponential complexity.

Let us now prove this proposition.

Proof. Firstly, let us remark that decipherment procedure $\boldsymbol{D}$ leaks only the following information:

- the activation value $\boldsymbol{V}$,
- the fact that the virus looks for the system time and date,
- the fact that the virus scans for specific data $\boldsymbol{\alpha}$, $\boldsymbol{\iota}$ and $\boldsymbol{\pi}$.

Moreover, the analyst is able to analyze the virus if and only if he knows the secret key $\boldsymbol{K_1}$. It can be obtained either by direct cryptanalysis or by guessing the exact values of the different activation data required to generate the good key. This guessing is equivalent to dictionary attacks. The cryptanalysis approach aims at finding the value $\boldsymbol{K_1}$ such that $\boldsymbol{H(K_1 \oplus C_1) = M}$ where $\boldsymbol{M}$ and $\boldsymbol{C_1}$ are easy to identify in the decipherment procedure $\boldsymbol{D}$. A hash function is highly non injective by nature. Thus it cannot be computationaly inversed in any way (*preimage resistance*). Consequently, this problem must be reformulated as a collision search problem (for more details, refer to (Menezes, 1997, chap. 9). In other words, find all pairs of input $\boldsymbol{x}$ and $\boldsymbol{x'}$ such that $\boldsymbol{H(x) = H(x')}$. This problem itself is computationally infeasible. To be more precise finding such a pair requires $\boldsymbol{2^{n/2}}$ operations for $\boldsymbol{n}$-bit input values ($\mathbf{n = 512}$ for SHA-1). Since the analyst must absolutely find the exact key $\boldsymbol{K_1}$ (secret

key really used to encrypt the viral code), he must beforehand compute all the values x such that $H(x) = M$. For a n-bit input, m-bit output hash function, there exists 2^{n-m} such x in average (2^{352} for SHA-1). Then, to summarize, recovering the key requires $2^{n/2} \times 2^{n-m}$ operations – that is to say $2^{(n \cdot n - m)/2}$ operations ($\approx 2^{131,072}$ for SHA-1).

Let us now consider the dictionary attack approach. It consists in enumerating all the possible values that might have been used as activation data. Note that, in that particular case, the analyst must simultaneously consider both the encrypted viral code and the system in which the code has been found. The analyst can try all the possible data relevant to the system (that is to say α, ι and δ) over which he has control during the analysis. Unfortunately, data π remains out of his control and thus he will not be able to determine its exact value. Thus there is no any other more efficient approach than searching exhaustively for the value $\alpha \oplus \delta \oplus \iota \oplus \pi$. Since at least π will be chosen randomly by the viral code author, this exhaustive search has complexity 2^n if a n-bit input hash function has been used (2^{512} for SHA-1). All things considered, the overall complexity of the code analysis is $\min(2^n, 2^{(n.n-m)/2}) = 2^n$. ■

Conclusion

The proof-of-concept virus bradley has been designed and discussed to illustrate the fact that efficient armouring is possible. BRADLEY and other efficient viruses of same kind pose the problem of a threat which so far, is impossible to deal with. The polymorphic nature of such codes, when optimally implemented, forbids any detection based only on the decipherment procedure D. During the experiments, detection based on behaviour monitoring and analysis has been successfully bypassed as well.

Permament and direct memory monitoring might be a solution to deal with such efficient armoured codes (use of *rootkit* or *honeypots*). This approach is only of theoretical interest since it requires a huge number of well-equipped computers in order to catch a viral copy with a significant probability of success. Besides, heavy system resources are required and this approach implies to be aware of this particular threat (efficient code armouring). Current research carried out in our laboratory aims at proving that even memory management and monitoring can be bypassed. We particularly designed a far more complex variant directly drawn from the *DarkParanoid* virus: at any time, only a single instruction can be found in an unencrypted form in memory. But it requires a far more complex environmental key generation than that simple one presented in Section 4.

Unless a solution is rapidly found to fight against these virus, this study outlines that an isolation of critical networks and a strict computer security policy is absolutely essential. Moreover, this implies that the antiviral companies must develop cryptanalysis skills in the very near future, under the assumption that it is possible to obtain a viral code sample and that breakable cryptosystems have been used.

References

Balepin I, (2003). Superworms and Cryptovirology: a Deadly Combination, from http://wwwcsif.cs.ucdavis.edu/~balepin/new_pubs/worms-cryptovirology.pdf.

Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S, Yang K. (2001), On the (Im)Possibility of Obfuscation Programs. In Advances in Cryptology, Crypto 2001, Lecture Notes in Computer Science 2139,1–18, Springer Verlag.

Chow S., Eisen P. Johnson H., Zakharov V.A. (2001), An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs.In Information Security, ISC 2001, Lecture Notes in Computer Science 2200, 144–155, Springer Verlag.

Ciubotariu M. (2003), Virus Cryptoanalysis, Virus Bulletin, november 2003. from http://www.virusbtn.com/magazine/archives/200311/cryptoanalysis.xml

Collberg C.S., Thomborson C. (2002), Watermarking, Tamper-proofing and Obfuscation - Tools for Software Protection, IEEE Transactions on Software Engineering, 28(8), August 2002, 735-746.

Ferrie P., Ször P. (2001), Zmist Opportunities, Virus Bulletin, March 2001, 6–7.

Filiol E. (2003), Les virus informatiques : théorie, pratique et applications, Collection IRIS, Springer. English translation to be published in March 2005 : Computer Viruses : from Theory to Applications, Springer Verlag.

Menezes A., van Oorschot P., Vanstone S.A. (1997), Handbook of Applied Cryptography, CRC Press.

National Institue of Standards and Technology, NIST FIPS PUB 180, Secure Hash Standard, U.S. Department of Commerce, May 1993.

Pearce S. (2003), Viral Polymorphism, SANS Institute.

Project funded by the Fund for Scientific Research - Flanders (2003), Coordinated Research of Program Obfuscation, from http://www.elis.regent.be/~banckaer/obfuscation/proposal.html

Riordan J., Schneier B. (1998) Environmental key generation towards clueless agents. In G. Vigna (Ed.) Mobile Agents and Security Conference'98, Lecture Notes in Computer Science, 15–24, Springer-Verlag.

Rivest R.L. (1992), The RC4 Encryption Algorithm, RSA Data Security Inc.

Rivest R.L., Robshaw M.J.B. Robshaw, Sidney R. and Yin Y.L. (1998), The RC6 block cipher. In Proceedings of the 1st AES Candidate Conference, August 1998, Ventura.

Shah P. (2002), Code Obfuscation For Prevention of Malicious Reverse Engineering Attacks, from http://islab.oregonstate.edu/koc/ece478/02Reports/S2.pdf

Wiley B. (2002), Curious Yellow: The First Coordinated Worm Design, from http://blanu.net/curious_yellow.html.

Young A, Yung M. (1996), Cryptovirology: Extorsion-Based Security Threats an Countermeasures, IEEE Symposium on Security and Privacy, Oakland, CA, 1996.

Young A, Yung M. (2004), Malicious Cryptography: Exposing Cryptovirology, Wiley & Sons. See also http://www.cryptovirology.com/