Lappeenranta University of Technology 5 April 2006

Department of Information Technology

TI5318800 Secured Communications
Seminar Work

# Polymorphic virus detection technology

Yevgeniy Bondarenko and Pavel Shterlayev, IT
contact: Yevgeniy.Bondarenko@lut.fi and Pavel.Shterlayev@lut.fi

# Contents

# Abstract

Viruses have become a big threat nowadays, however there plenty of software products which can be suggested as protection tools. The aim of our paper is to describe briefly the way viruses work in general, giving emphasis to polymorphic viruses. Several algorithms of protection are defined in our paper to give a full overview of a problem. Furthermore, we tried to depict the details of antivirus work, using modern algorithms, like Stryker, produced by Semantics. For better performance, AVZ antivirus product is used, as very factual tool, designed for professionals.

# 1. Introduction

Polymorphic computer viruses are the most complex and difficult viruses to detect, often requiring antivirus companies to spend days or months creating the detection routines needed to catch a single polymorphic.

This paper provides an overview of polymorphics and existing methods of detection.

# 2. The Evolution of Polymorphic Viruses

A computer virus is a self-replicating computer program that operates without the consent of the user. It spreads by attaching a copy of itself to some part of a program file, such as a spreadsheet or word processor. Viruses also attack boot records and master boot records, which contain the information a computer uses to start up. Macro viruses attack such files as word processing documents or spreadsheets.

Most viruses simply replicate. Some display messages. Some, however, deliver a payload — a portion of the virus program that is designed to corrupt programs, delete files, reformat a hard disk, or crash a corporate-wide network, potentially wiping out years of data and destroying critical information.

### a) Simple Viruses

A simple virus that merely replicates itself is the easiest to detect. If a user launches an infected program, the virus gains control of the computer and attaches a copy of itself to another program file. After it spreads, the virus transfers control back to the host program, which functions normally.

Yet no matter how many times a simple virus infects a new file or floppy disk, for example, the infection always makes an exact copy of itself. Antivirus software need only search, or scan, for a defined sequence of bytes — known as a signature — found in the virus.

### b) Encrypted Viruses

In response, virus authors began encrypting viruses. The idea was to hide the fixed signature by scrambling the virus, making it unrecognizable to a virus scanner.

From Figure 2 to Figure 4 the process of virus decryption and execution is clearly shown, using decryption key (Figure 1).

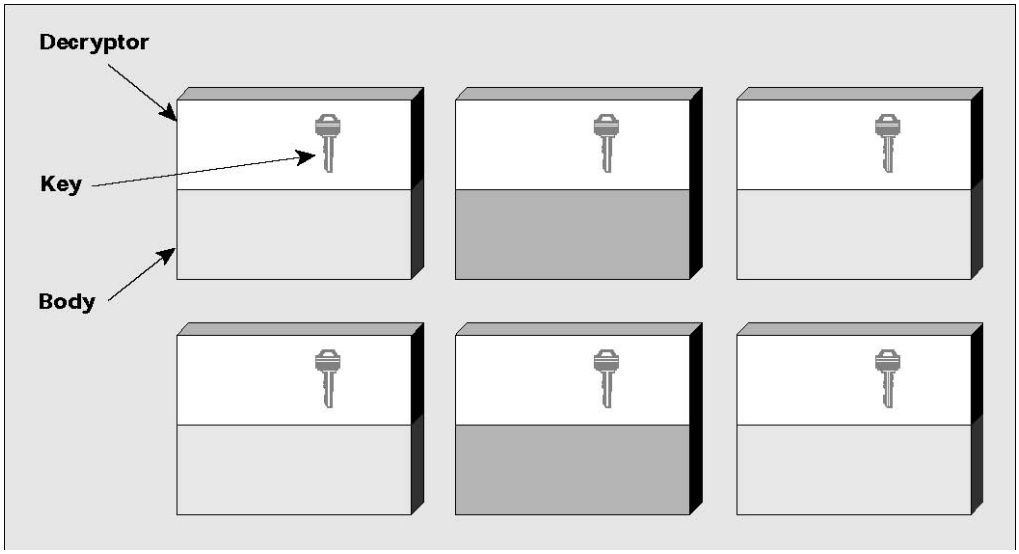


**Figure 1.** An encrypting virus always propagates using the same decryption routine. However, the key value within the decryption routine changes from infection to infection. Consequently, the encrypted body of the virus also varies, depending on the key value.

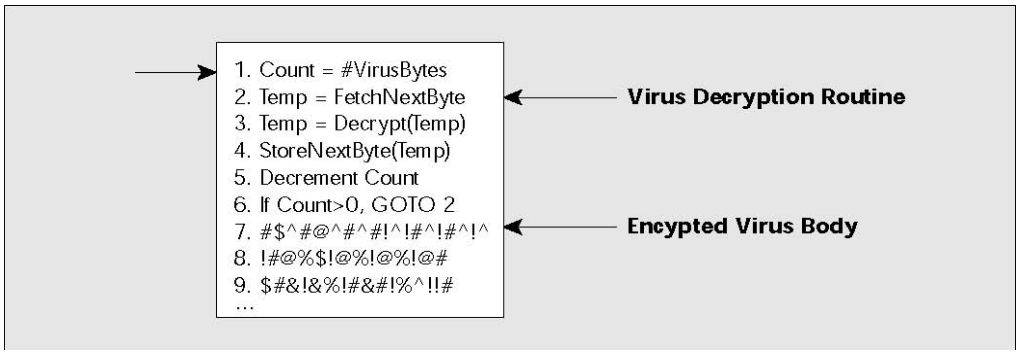


**Figure 2.** This is what an encrypted virus looks like before execution

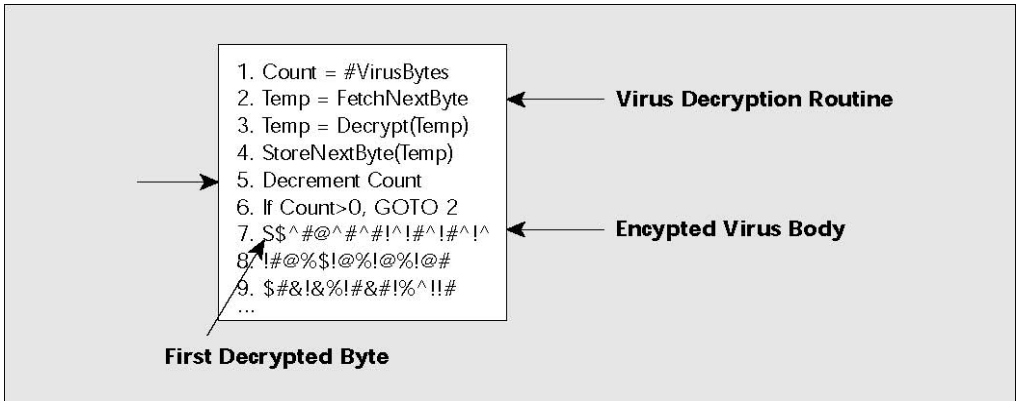


**Figure 3.** At this point, the virus has executed its first five instructions and has decrypted the first byte of the encrypted virus body

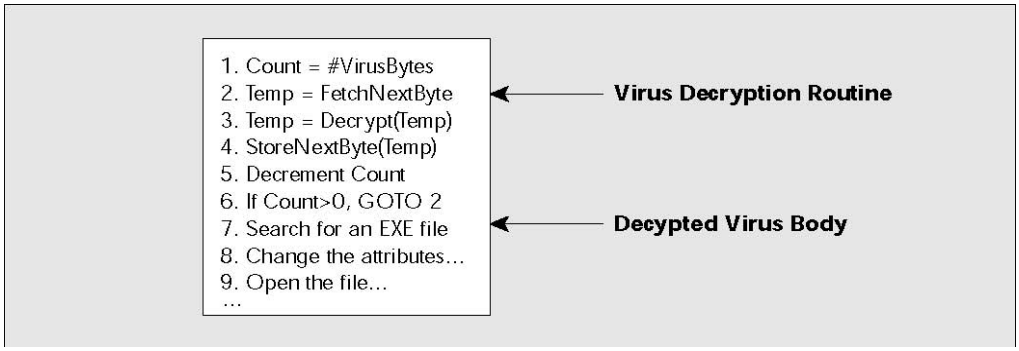


**Figure 4.** This is the fully decrypted virus code.

An encrypted virus consists of a virus decryption routine and an encrypted virus body. If a user launches an infected program, the virus decryption routine first gains control of the computer, then decrypts the virus body. Next, the decryption routine transfers control of the computer to the decrypted virus.

An encrypted virus infects programs and files as any simple virus does. Each time it infects a new program, the virus makes a copy of both the decrypted virus body and its related decryption routine, encrypts the copy, and attaches both to a target.

To encrypt the copy of the virus body, an encrypted virus uses an encryption key that the virus is programmed to change from infection to infection. As this key changes, the scrambling of the virus body changes, making the virus appear different from infection to infection. This makes it extremely difficult for antivirus software to search for a virus signature extracted from a consistent virus body.

However, the decryption routines remain constant from generation to generation — a weakness that antivirus software quickly evolved to exploit. Instead of scanning just for virus signatures, virus scanners were modified to also search for the signature that identified a specific decryption routine.

### c) Polymorphic Viruses

In retaliation, virus authors developed the polymorphic virus. Like an encrypted virus, a polymorphic virus includes a scrambled virus body and a decryption routine that first gains control of the computer then decrypts the virus body.

However, a polymorphic virus adds to these two components a third — a mutation engine that generates randomized decryption routines that change each time a virus infects a new program.

In a polymorphic virus, the mutation engine and virus body are both encrypted. When a user runs a program infected with a polymorphic virus, the decryption routine first gains control of the computer, then decrypts both the virus body and the mutation engine. Then, the decryption routine transfers control of the computer to the virus, which locates a new program to infect.

The virus makes a copy of both itself and the mutation engine in random access memory (RAM). The virus then invokes the mutation engine, which randomly generates a new decryption routine that is capable of decrypting the virus, yet bears little or no resemblance to any prior decryption routine. Next, the virus encrypts this new copy of the virus body and mutation engine. Finally,

the virus appends this new decryption routine, along with the newly encrypted virus and mutation engine, onto a new program.

As a result, not only is the virus body encrypted, but the virus decryption routine varies from infection to infection. This confounds a virus scanner searching for the defined sequence of bytes that identifies a specific decryption routine.

With no fixed signature to scan for, and no fixed decryption routine, no two infections look alike. The result is a formidable adversary.

## 3. The Scale of the Problem

The Tequila and Maltese Amoeba viruses caused the first widespread polymorphic infections in 1991.

In 1992, in a harrowing development, Dark Avenger, author of Maltese Amoeba, distributed the Mutation Engine, also known as MtE, to other virus authors with instructions on how to use it to build still more polymorphics.

It is now common practice for virus authors to distribute their mutation engines, making them widely available for other virus authors to use as if they were do-it-yourself kits.

Today, antivirus researchers report that polymorphic viruses comprise about five percent of the more than 8,000 known viruses.

Of these, SARC reports only a small number of polymorphics “in the wild” — just 20 as of mid-1996. Yet this represents an increase of 25 percent from 16 polymorphics in the wild in mid-1995, a year earlier. Also, antivirus researchers have identified 50 mutation engines. SARC reports 13 mutation engines in the wild as of mid-1996, up 30 percent in one year from 10 mutation engines reported in the wild as of mid-1995.

Two polymorphics — One Half and Natas — rank among the 20 most-prevalent computer viruses, according to the *1996 Computer Virus Prevalence Survey* conducted by the National Computer Security Association (NCSA).

One Half slowly encrypts a hard disk. Natas, also known as SatanBug.Natas, is highly polymorphic, designed to evade and attack antivirus software. It infects .COM and .EXE program files.

## 4. Methods of viruses detection

In general, all antivirus algorithms, which are used for computer virus detection, are divided in four major groups:

a. Using Cyclical Redundancy Check (CRC) for a check of file entire.
b. Signature search.
c. Logic search.
d. Spying of process behavior.

### a) Using CRC for a check of file entire

This method is quite old and widely used in most antivirus software, because it is still very reliable. However, it has a several disadvantages as well as advantages.

- **Advantages are:**

Simple realization of the algorithm, -

This algorithm creates special tables for storing files CRC sum. It goes through a folders tree and special tables simultaneously, if it finds new unregistered file antivirus[1] calculates its CRC sum and saves it in these tables, if file is already registered in the tables. Then antivirus compares its CRC value with their previously calculated value.

High level of virus detection, -

This algorithms group works very good even in those cases, when virus doesn't change files size, and virus structure and newness doesn't play significant role.

- **Disadvantages are:**

Usage of special tables,-

They can be attacked (changed or removed by viruses), between scanning sessions.

It is possible that mechanism of virus and hardware interaction operates on lower level, than antivirus software does.

In instance, antivirus controls files with MS-DOS interruptions, while a virus occupies "int 13h", and gives all data to antivirus, which antivirus doesn't see suspicious.

It can't define a number of active viruses.

It can't define types of viruses.

It has poor probability of automatic cure.

Special tables can store additional information about files, which then will be used to recover them, after attack. But if virus uses file encryption or just changes victim body very much, then this information wouldn't be enough for restoring of the infected files.

Process of scanning takes huge amount of time.

Nowadays, Hard Disk Drivers (HDD) have big capacity. So, even one scan of a full space will take long time. Of course, modern operation system (OS) have multitasking issue, but if a system has only one processor, every process will take a little piece of processor's time. When there are lots of processes in a system, then this system will slow down dramatically. So, antivirus work, even, in background mode can annoy users. This gives viruses an opportunity to spy a run of such antivirus software and "help" them to reduce productivity of the system.

Data can be infected between scanning sessions.

This algorithm can be used only for executable files protection.

CRC can't be used to provide virus detection for documents, scripts or configuration files which can be used and modified by users quite often, like MS Word documents, i.e. these algorithms can't defend users from internet-worms and Trojans. Furthermore, a virus can save its body in *.zip file (and etc.) and falsify its header and then execute it in wide range of ways (autorun).

If we draw a conclusion, these types of anti-viruses have become out of a date. Firstly, in our days, people exchange executable files very rare, but documents, archives, and so on instead.

[1] Here and further in this passage one should comprehend antivirus as antivirus, which uses certain algorithm of corresponding chapter

Despite this, this technology can be used in combination with other algorithms of virus searching quite successfully.

**b) Signature search**

Signature search belongs both to polyphage-antivirus and scan-antivirus software. The difference is that polyphage-antivirus searches and cures all viruses that it is used to know, and scan-antivirus tries to find all forms that seem suitable for viruses. Both of them use signatures in their work – sequences of commands, typical for viruses and Trojan components.

Algorithm of realization is quite simple, as in previous case:

If we would simplify, it searches for a file, then looks for a first byte of a signature and compares whole file block to this signature. If it finds their equality, it will alarm, and then continue.

Heuristic signatures can be referred to:

- Searching of files using mask like “*.com”, “*.exe”.
- Getting of files attributes
- Getting of the time of last file modification
- Opening file for reading/writing
- Reading from file
- Writing to file
- Closing of file
- Recovering of time of last file modification
- Restoring of file attributes
- Requesting for leaving part of program in resident memory
- Writing to system areas of operation system

For decision making about virus detection, signature search uses amount of signatures and their weight that has been found in a certain file. Certain virus can have common areas in all its variations. So, polyphages can also use CRC calculation of these certain areas in a file, for better confidence. If virus uses quite simple encryption algorithm for hiding of its body, signature search uses decryption algorithms firstly. In more complicated cases, when virus adds random code to its body, this algorithm uses search with floatable mask procedures. Also, most scanners try to search not for exact signature, but very near to it.

- **Advantages of this algorithm are:**

  High level of viruses and their “culture” detection.

  Also, it provides high probability of unknown virus detection.

  Acceptable speed of work

  It allows user to set level of security

  Possibility to load new signatures and data concerning new viruses rapidly

- **Disadvantages are:**

Main shortcoming is if the algorithm of virus encryption is changed, even insignificant, the algorithm of signature searching would require huge modifications.

Mixing of commands and blocks of a virus is usual, but requires huge modification of searching algorithm. If virus encrypts its main body with static algorithm, and uses non-encrypted area (decrypt mechanism), then step-by-step execution of commands is needed for its detection and cure. Step-by-step execution of commands can be done in two ways: tracing and emulation. However, tracing is unreliable and process of emulation is complicated and time-consuming. In case, when virus doesn't encrypt its body, but generates it completely, it's required to develop special algorithms, which can solve "inverse" tasks. It means, rearranging commands of a virus or removing blocks of virus commands, to standard templates i.e. it simply converts virus body to some standard virus in working state.

Polymorphic virus detection need very complicated search procedure.

In polymorphic viruses constant bytes can be absent, and encryption algorithm can be quite difficult. Also, decryption key may change in every single case.

It requires deactivation of resident viruses before scan.

If resident viruses wouldn't be deactivated before scan, then resident virus could infect files during checking. Furthermore, these viruses can hide themselves in checking files. Therefore, antivirus has to make memory check before proceeding and do everything to neutralize resident programs, while checking.

Only limited number of file types can be cured.

Signature search can be possibly used for checking any types of files, but if your target is to cure damaged file antivirus should have knowledge about format of the file. This is huge obstacle for universality of polyphage. That's why additional modules for protection are produced for individual software (MS Word, MS Outlook).

Useless signatures can be turned off in order to increase productivity. Therefore, antivirus checks file extension before scanning and chooses signatures, which are appropriate in this case. However, it can be easily used through falsifying of file header.

Data can be infected between scanning sessions.

Increase of security level leads to increase of false alarms.

**c) Logical search**

This is the most modern algorithm of viruses search. The main idea of the algorithm is an execution of files it is going to check inside a virtual machine. The difference from signature search concludes in that it checks state of virtual machine, instead of files.

The easiest example about how simple set of commands or instructions can lead to a same state:

```
Mov AX,0
Sub AX, AX
XOR AX, AX
MOV BX, 0 / XCHG BX, AX
Xor BX, BX / PUSH BX / POP AX
```

```
Mov BX, AX / XOR AX, BX
```

As one can understand, all these sets of commands (one line per set, "/" divide off commands) result in that register AX equals to zero. And so, antivirus checks only final state, rather than checking correspondence of virus commands with enormous number of instruction's set, which put in this final state. This approach can catch both known viruses and new super polymorph viruses.

- **Advantages of this method are:**

Algorithm of realization can be quite simple.

In this case, algorithm has big productivity but can result in lower level of information it can distribute to the user.

Algorithm of realization can be very complicated.

In this case, neuronet can be used. This approach is opposite to previous one, resulting in higher universality of this method.

High level of viruses and Trojan components detection.

Allows its usage in real-time applications.

- **Disadvantages of this method are:**

Algorithm can't check a file without its whole execution.

There are some viruses, which executes some of their parts depending on some external events. That's why antivirus should try to give management to virus procedures in all possible cases. But it's impossible, because we can define enormous number of cases. Virus procedure can be put in waiting state listening for existence of a file in certain place of HDD, while infected program is working.

Also, antivirus can check segment of code of suspicious file, which contains small procedure, which waits for some event happening, to decrypt virus. This analysis will make the algorithm much more complicated, but even this can't provide absolute result. Example of event:

If algorithm finds RAR archievator on a disk, then "C" virus branch of algorithm is started. This "C" branch can decrypt virus body using RAR with some unknown key. Process of password search can consume much time. This is acceptable for virus, but not for antivirus. Moreover, algorithm of logical search can't be used to search for a virus in certain program. In event, that was previously described, virus executes additional program (RAR), which also can execute other processes. So, process of scanning can go to infinity, because time interval, when program operates can take much time, and number of processes like this unlimited.

Let's consider that a virus consists of a several independent parts:

- Resident;
- Password search procedure;
- Main body of virus and victim.

In this case, event waiting procedure (RAR existence) splits virus into different components: resident, password search procedure. The resident gets management of the process, while parent process is terminated. Password search procedure takes certain blocks of the virus from main body of victim and starts do decrypt them. The resident is waiting for termination of password search procedure, and then executes the virus. This situation can be more complicated, if event waiting

procedure would be mixed up with a victim's body and virus would use its calculations. This leads to curing troubles, because it's quite difficult to distinguish victim and virus in this case.

Anyway, this is the only one way of virus pursuit, because searching by signature is useless.

Algorithm can't distinguish program and virus operations.

It was shown in previous example.

**d) Spying of process behavior.**

There exist some antiviruses, which block behavior of a program. Algorithm works as following:

It monitors a system in a real-time mode, looking for process requests to the system, which allows them to access into files. Then, it should check whether this process has permission to access or not, and if there is no note about this process in a data base, algorithm can ask for a users permission.

This type of antiviruses has been changed dramatically during last few years. Nowadays, they are automated completely. They import database with settings, which contain list of files. This list defines with which files certain software can operate.

This algorithm is used for user data protection very rarely. Basically, it used for operation system files protection and files of antivirus itself. Recently, new application technology was proposed. The main idea of this technology is to litter up a disk with file-traps and to control an access to these files.

- **Advantages of this method are:**

High level of processing speed.

Antivirus doesn't need to control all processes in a system. Suspicious process is caught while it tries to get access to file-trap.

Antivirus doesn't need to have an ability to work with all file formats.

File-traps can be positioned in places, where files with the same extension are placed. These file-traps look like chameleons, imitating a state of surroundings.

- **Disadvantages:**

The number of file-traps can be quite big, littering up the space on HDD.

It doesn't provide one hundred percent of files control.

It can't indicate the reasons.

According to this algorithm, a process, which tries to get access to a file-trap, is killed by an antivirus. Antivirus doesn't investigate any reasons, which caused this process behavior. So, we can define a situation, when before infecting a file, a virus can try to execute this file with some casual software, such as Notepad, MS Word and so forth. If execution succeeds, then the virus kills this process and infects the file. If fails, virus can try to infect next file. So, antivirus logs will contain information about Notepad (if file-trap was run by Notepad) and software like this as an infected, but it's not a truth.

# 5. Polymorphic Detection

Antivirus researchers first fought back by creating special detection routines designed to catch each polymorphic virus, one by one. By hand, line by line, they wrote special programs designed to detect various sequences of computer code known to be used by a given mutation engine to decrypt a virus body i.e. signature search was adopted to this task.

Unfortunately, this approach proved inherently impractical, time-consuming, and costly. Each new polymorphic requires its own detection program. Also, a mutation engine produces seemingly random programs, any of which can properly perform decryption — and some mutation engines generate billions upon billions of variations.

Moreover, many polymorphics use the same mutation engine. Also, different engines used by different polymorphics often generate similar decryption routines, which make any identification based solely on decryption routines wholly unreliable.

This approach also leads to mistakenly identifying one polymorphic as another. These shortcomings led antivirus researchers to develop generic decryption techniques that trick a polymorphic virus into decrypting and revealing itself.

### a) Generic Decryption

Generic decryption assumes:

- The body of a polymorphic virus is encrypted to avoid detection.
- A polymorphic virus must decrypt before it can execute normally.
- Once an infected program begins to execute, a polymorphic virus must immediately usurp control of the computer to decrypt the virus body, and then yield control of the computer to the decrypted virus.

A scanner that uses generic decryption relies on this behavior to detect polymorphics. Each time it scans a new program file, it loads this file into a self-contained virtual computer created from RAM. Inside this virtual computer, program files execute as if running on a real computer.

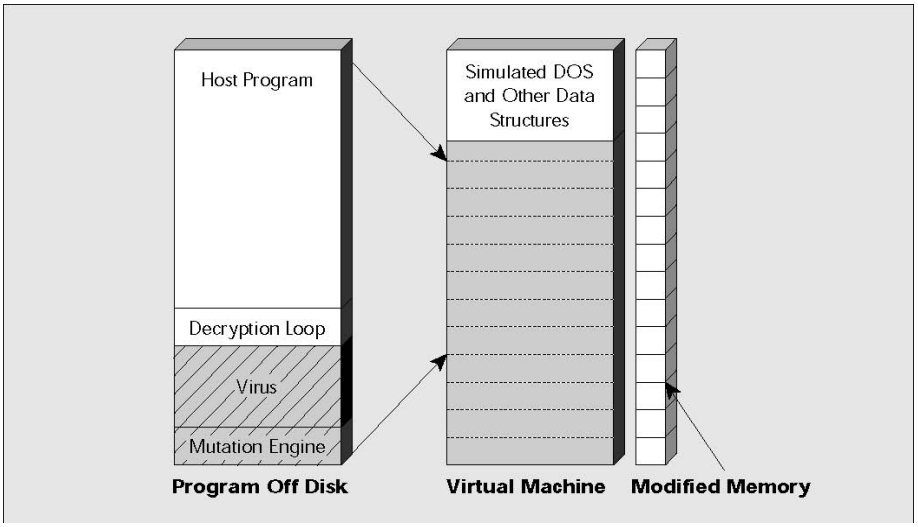


**Figure 5.** The generic decryption engine is about to scan a new infected program

The scanner monitors and controls the program file as it executes inside the virtual computer (see Figure 6). A polymorphic virus running inside the virtual computer can do no damage because it is isolated from the real computer.

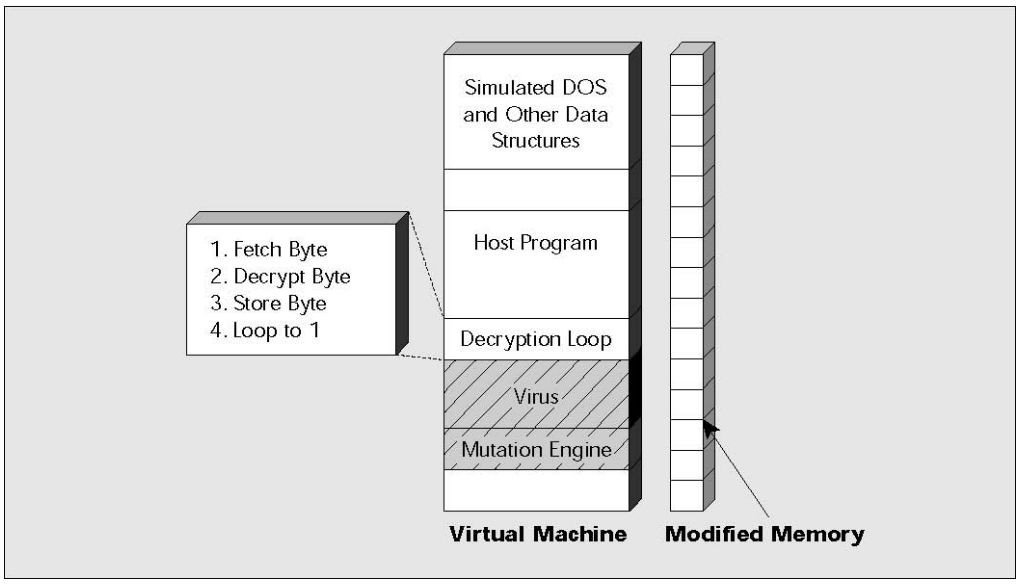


**Figure 6.** The generic decryptor loads the next program to scan into the virtual machine. Notice that each section of memory in the virtual machine has a corresponding modified memory cell depicted on the right-hand side of the virtual machine. The generic decryption engine uses this to represent areas of memory that are modified during the decryption process.

When a scanner loads a file infected by a polymorphic virus into this virtual computer, the virus decryption routine executes and decrypts the encrypted virus body. This exposes the virus body to the scanner, which can then search for signatures in the virus body that precisely identify the virus strain.

If the scanner loads a file that is not infected, there is no virus to expose and monitor. In response to nonvirus behavior, the scanner quickly stops running the file inside the virtual computer, removes the file from the virtual computer, and proceeds to scan the next file.

The process is like injecting a mouse with a serum that may or may not contain a virus, and then observing the mouse for adverse affects. If the mouse becomes ill, researchers observe the visible symptoms, match them to known symptoms, and identify the virus. If the mouse remains healthy, researchers select another vial of serum and repeat the process.

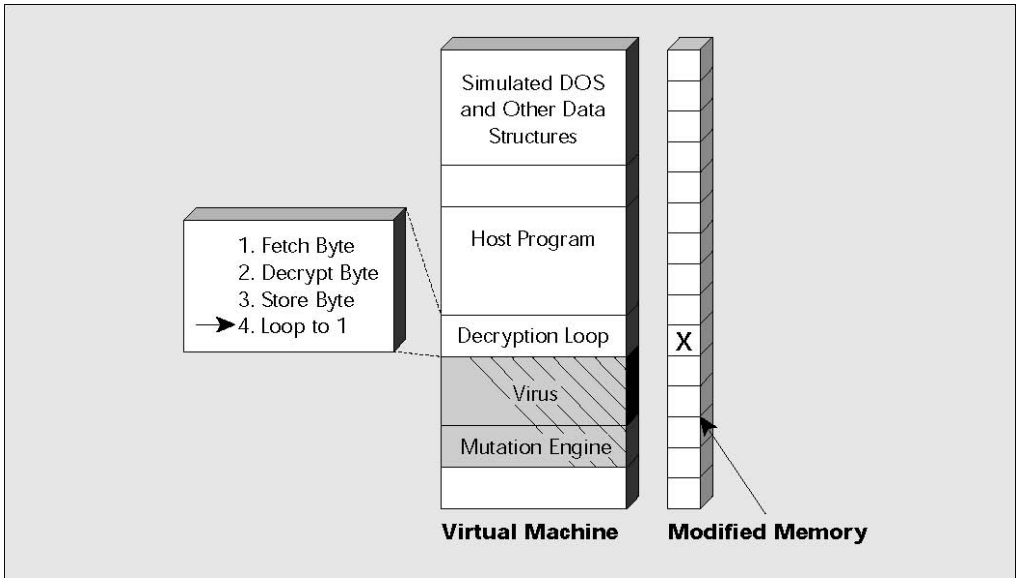


**Figure 7.** At this point the generic decryption engine passes control of the virtual machine to the virus and the virus begins to execute a simple decryption routine. As the virus decrypts itself, the modified memory table is updated to reflect the changes to virtual memory

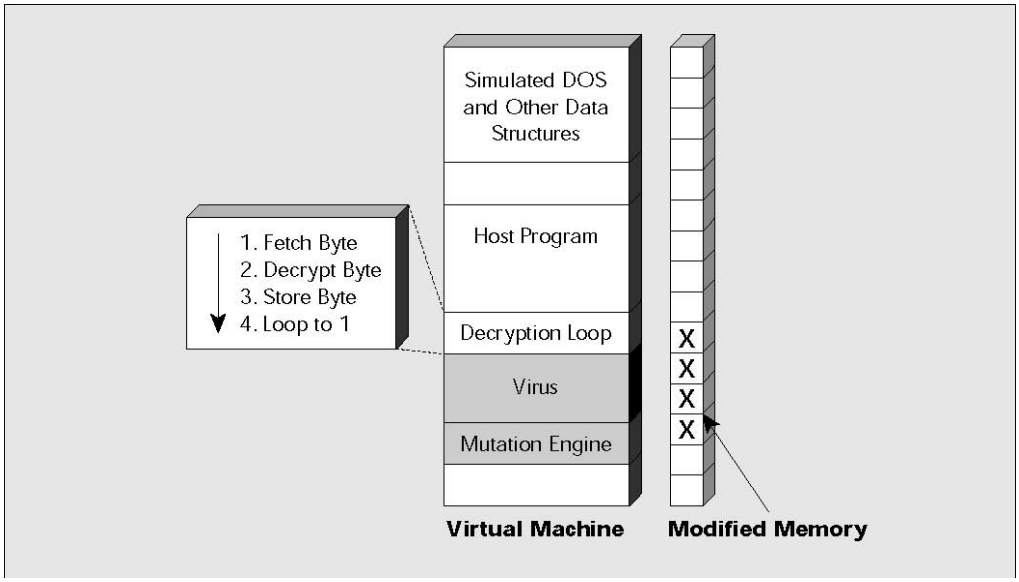


**Figure 8.** Once the virus has decrypted enough of itself, the generic decryption engine advances to the next stage

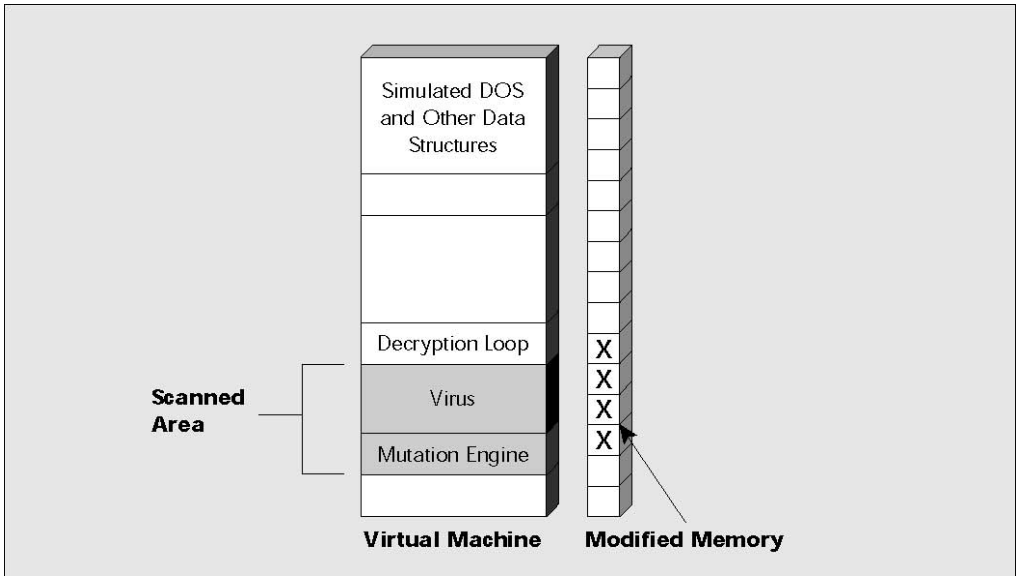


**Figure 9.** Now the generic decryption scanner searches for virus signatures in those areas of virtual memory that were decrypted and/or modified in any way by the virus. This is the most likely location for virus signatures

The key problem with generic decryption is speed. Generic decryption is of no practical use if it spends five hours waiting for a polymorphic virus to decrypt inside the virtual computer. Similarly, if generic decryption simply stops short, it may miss a polymorphic before it is able to reveal enough of itself for the scanner to detect a signature.

**b) Heuristic-Based Generic Decryption**

To solve this problem, generic decryption employs "heuristics," a generic set of rules that helps differentiate non-virus from virus behavior.

As an example, a typical nonvirus program will in all likelihood use the results from math computations it makes as it runs inside the virtual computer. On the other hand, a polymorphic virus may perform similar computations, yet throw away the results because those results are irrelevant to the virus. In fact, a polymorphic may perform such computations solely to look like a clean program in an attempt to elude the virus scanner.

Heuristic-based generic decryption looks for such inconsistent behavior. An inconsistency increases the likelihood of infection and prompts a scanner that relies on heuristic-based rules to extend the length of time a suspect file executes inside the virtual computer, giving a potentially infected file enough time to decrypt itself and expose a lurking virus.

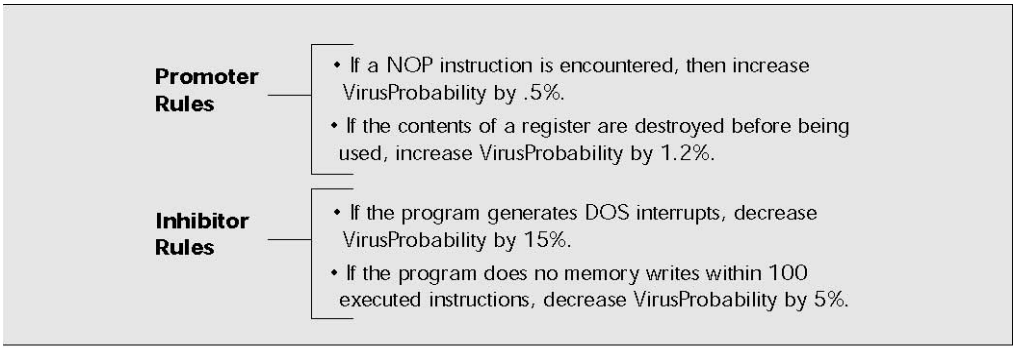


**Figure 10.** Initially the generic decryptor assumes that every has 10% probability of infection. Emulation continues as long as the virus probability is greater than zero. This virus probability is updated as the various rules observe virus-like or non-virus-like behavior during emulation.

Unfortunately, heuristics demand continual research and updating. Heuristic rules tuned to detect 500 viruses, for example, may miss 10 of those viruses when altered to detect 5 new viruses.

Also, as virus writers continue trying to make viruses look like clean programs, heuristics can easily balloon to the point where almost any program might share attributes that trigger the scanner to lengthen the time it takes to examine a file.

In addition, generic decryption must rely on a team of antivirus researchers able to analyze millions of potential virus variations, extract a signature, then modify a set of heuristics while also guarding against the implications of changing any heuristic rules. This requires extensive, exhaustive regression testing. Without this commitment, heuristics quickly becomes obsolete, inaccurate, and inefficient.

### c) The Striker System

Like generic decryption, each time it scans a new program file, Striker loads this file into a self-contained virtual computer created from RAM. The program executes in this virtual computer as if it were running on a real computer.

However, Striker doesn't rely on heuristic guesses to guide decryption. Instead, it relies on virus profiles or rules that are specific to each virus, not a generic set of rules that differentiate non-virus from virus behavior.

When scanning a new file, Striker first attempts to exclude as many viruses as possible from consideration, just as a doctor rules out the possibility of chicken pox if an examination fails to detect scabs on a patient's body.

For example, different viruses infect different executable file formats. Some infect only .COM files. Others infect only .EXE files. Some viruses infect both. Very few infect .SYS files. As a result, as it scans an .EXE file, Striker ignores polymorphics that infect only .COM and .SYS files. If all viruses are eliminated from consideration, then the file is deemed clean. Striker closes it and advances to scan the next file.

If this preliminary scan doesn't rule out infection, Striker continues to run the file inside the virtual computer as long as the behavior of the suspect file is consistent with at least one known polymorphic or mutation engine.

For example, one polymorphic virus is known to perform math computations and throw away the results. A second polymorphic may never perform such calculations. Instead, it may use specific random instructions in its decryption routine. A third polymorphic may call on the operating system as it decrypts.

Striker catalogs these and nearly 500 other characteristics into each virus profile, one for each polymorphic and mutation engine.

Consider a set of generic heuristic rules that identify A, B, C, D, and E as potential virus behaviors. In contrast, a Striker profile calls for virus 1 to execute behaviors A, B, and C. As it decrypts, virus 2 executes behaviors A, B, and D, while Virus 3 executes behaviors B, D, and E. If Striker observes behavior A while running a suspect file inside the virtual computer, this is consistent with viruses 1 and 2. However, it is not consistent with Virus 3. Striker eliminates Virus 3 from consideration.

The heuristic-based system must continue searching for all three viruses, however, because it observes behavior that is consistent with its generic rules.

If Striker next observes behavior B, this is consistent with viruses 1 and 2. Striker must continue scanning for these two viruses. However, the heuristics again continue to search for all three viruses.

Finally, if Striker observes behavior E, this eliminates Virus 2 from consideration, and Striker now pursues a single potential virus.

The heuristic-based scanner continues to search for all three viruses.

Under Striker, this process continues until the behavior of the program running inside the virtual computer is inconsistent with the behavior of any known polymorphic or mutation engine. At this point, Striker excludes all viruses from consideration.

On the other hand, a heuristic-based system scans for all viruses all the time. It must find some behavior inconsistent with all behaviors.

- **Striker's Strategic Advantages**

Clearly the first advantage to Striker's approach is speed. The profiles enable Striker to quickly exclude some polymorphic viruses and home in on others. In contrast, heuristics labor on, scanning all program files against all available generic rules of how all known polymorphics and all known mutation engines might behave.

The profiles also enable Striker to process uninfected files quickly, minimizing impact on system performance. In contrast, heuristic-based scanning is more likely to decrease system performance, because uninfected files must also be scanned against all generic rules for how all known polymorphics and mutation engines might behave.

Second, antivirus researchers are no longer forced to rewrite complex heuristic rules to scan for each new virus, then exhaustively test and retest to ensure they do not inadvertently miss a polymorphic the software previously detected.

Third, with Striker, a team of antivirus researchers may work in parallel, building profiles for many new polymorphic viruses, swiftly adding each to Striker. Each profile is unique, much like a virus signature, independent of any other profile. The old profiles still work, and the new profile

does not affect the old. Exhaustive, time-consuming regression testing is no longer necessary. It becomes easy to update antivirus software by compiling new virus profiles into antivirus database file that is posted online monthly or obtained on floppy disk.

# 6. Outlook

To date, generic decryption has proved to be the single most effective method of detecting polymorphics. Striker improves on this approach.

Yet it is only a matter of time before virus authors design some new, insidious type of virus that evades current methods of detection.

Virus authors might design a polymorphic virus that decrypts half the time, for example, yet remains dormant at other times. Antivirus software could not reliably detect such a virus if it does not decrypt itself every time the file is loaded into the virtual computer. In this case, a hand-coded detection routine will be needed.

Or, imagine a host program that waits for the user to press a specific key and then terminates. A polymorphic infecting this host might only take control just after the user enters the required key-stroke. If the user enters the keystroke, the virus runs. If not, the virus gets no opportunity to launch. However, inside the virtual computer created by generic decryption, the program would never receive the needed keystroke — and the virus would never have a chance to decrypt.

A small number of viruses are already resistant to detection by generic decryption. There’s no doubt that virus authors will continue to design new viruses, using new technologies, creating new problems. Antivirus researchers will need to deal with these new threats, just as Striker today delivers the solution that best protects computer users against polymorphics.

# AVZ antivirus program description

AVZ antivirus program was meant for curing and removal of:

- SpyWare and AdWare programs and modules;
- Network and mail worms;
- Trojan programs (including all their types likeTrojan-PSW, Trojan-Downloader, Trojan-Spy) and Backdoor(programs, which are designed for remote control of PC);
- Trojan Dialing programs (Dialer, Trojan.Dialer);
- Keyboard spies, and programs, which can somehow spy on the user.

Program can be concerned as an analogue to Trojan Hunter and LavaSoft Ad-aware 6.Primary target of the program is removal of SpyWare, AdWare and Trojan programs.

One of the characteristic properties of the program is possibility of putting settings for each kind of these harmful programs. For instance, we can set a removal mode for Trojans, but blocking mode for AdWare.

Another difference of AVZ is huge number of heuristic searches of the system, which don't depend on signature search. These searches are trying to find a RootKit, keyboard spies, various Backdoor with usage of typical TCP/UDP ports base. This type of search can manage to detect new virus types.

Besides the typical for such type of programs signature search, AVZ has tens thousand of digital signatures for system files. Usage of this base decreases a number of false abrasions of heuristics and helps to solve few tasks. For example, in a file searching system there is a filter, which excludes known files from search results, or if files should be moved into quarantine mode, blocking of known safe files is done.

Program limitations:

- As primary target of the program is SpyWare, AdWare and Trojan programs, it doesn't manage several types of archives, documents check;
- It doesn't deal with infected programs cure. For qualitative and safety curing special antiviruses should be used.

Utilities which are built into:

- Processes controller
- Drivers and services controller
- Log data search
- Search for files on a disk
- SPI/LSP manager
- Open TCP/UDP ports
- Autorun manager
- IE extensions manager

- Task Scheduler manager
- Explorer extensions manager
- Inculcated .dll manager
- Hosts file manager
- Printing system extensions manager
- Common resources and network sessions.

# Protocol of antivirus utility AVZ version 4.15

Scanning started at 05.04.2006 0:00:29
Base downloaded: 21076 signatures, 2 neuroprofiles, 55 curing micro programs, base is defined by 24.03.2006 17:30
Heuristic micro programs downloaded: 357
Digital signatures of system files downloaded: 48167
Heuristic Analisator Mode: Medium heuristics level
Curing Mode: Activated
1. Search for RootKit and programs, which catches API functions
1.1 Search for API interceptors API, which work in UserMode
 Analysis of kernel32.dll, export table was found in .text section
Function kernel32.dll:LoadLibraryA (578) is intercepted, method ProcAddressHijack.GetProcAddress ->7C882FC4<>7C801D77
Function kernel32.dll:LoadLibraryExA (579) is intercepted, method ProcAddressHijack.GetProcAddress ->7C882FD3<>7C801D4F
Function kernel32.dll:LoadLibraryExW (580) is intercepted, method ProcAddressHijack.GetProcAddress ->7C882FF1<>7C801AF1
Function kernel32.dll:LoadLibraryW (581) is intercepted, method ProcAddressHijack.GetProcAddress ->7C882FE2<>7C80ACD3
 Analysis of ntdll.dll, export table was found in .text section
 Analysis of user32.dll, export table was found in .text section
 Analysis of advapi32.dll, export table was found in .text section
 Analysis of ws2_32.dll, export table was found in .text section
 Analysis of wininet.dll, export table was found in .text section
 Analysis of rasapi32.dll, export table was found in .text section
 Analysis of urlmon.dll, export table was found in .text section
 Analysis of netapi32.dll, export table was found in .text section
1.2 Search for API interceptors API, which work in KernelMode
 Driver was downloaded successfully
 SDT was found (RVA=07B180)
 Kernel ntkrnlpa.exe in memory address 804D7000
   SDT = 80552180

KiST = 81609008 (297)
>>> Warning, KiST table was removed ! (80501030(284)->81609008(297))
Function ZwClose (19) is intercepted (805B06F8->F37F70D0), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwCreateKey (29) is intercepted (80618BDA->F854BFE0), interceptor a347bus.sys
Function ZwCreatePagingFile (2D) is intercepted (8059F8DE->F853FB00), interceptor a347bus.sys
Function ZwCreateProcess (2F) is intercepted (805C5CC6->F37F6D90), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwCreateProcessEx (30) is intercepted (805C5C10->F37F6F30), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwCreateSection (32) is intercepted (8059F222->F37F7210), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwCreateThread (35) is intercepted (805C5AAE->F37F7A50), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwEnumerateKey (47) is intercepted (8061941A->F85405DC), interceptor a347bus.sys
Function ZwEnumerateValueKey (49) is intercepted (80619684->F854C120), interceptor a347bus.sys
Function ZwOpenFile (74) is intercepted (8056E254->F890023E), interceptor kl1.sys
Function ZwOpenKey (77) is intercepted (80619F70->F854BFA4), interceptor a347bus.sys
Function ZwOpenProcess (7A) is intercepted (805BFB56->F37F67F0), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwQueryInformationFile (97) is intercepted (8056EB00->F37F7710), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwQueryKey (A0) is intercepted (8061A294->F85405FC), interceptor a347bus.sys
Function ZwQuerySystemInformation (AD) is intercepted (80606338->F37F7850), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwQueryValueKey (B1) is intercepted (80616C94->F854C076), interceptor a347bus.sys
Function ZwResumeThread (CE) is intercepted (805C949E->F37F7A00), interceptor C:\WINDOWS\System32\drivers\klif.sys
Function ZwSetSystemPowerState (F1) is intercepted (80646DE8->F854B550), interceptor a347bus.sys

Function ZwSuspendThread (FE) is intercepted (805C93D8->F37F79B0), interceptor C:\WINDOWS\System32\drivers\klif.sysFunction ZwTerminateProcess (101) is intercepted (805C74A6->F3A20320), interceptor C:\Program Files\Agnitum\Outpost Firewall\kernel\FILTNT.SYS
Function ZwWriteVirtualMemory (115) is intercepted (805A82DA->F3A20280), interceptor C:\Program Files\Agnitum\Outpost Firewall\kernel\FILTNT.SYS
Number of functions that were checked: 284, were intercepted: 21, reconstructed: 0
2. Memory check
Number of processes found: 54
Number of downloaded modules: 450
Memory check is completed
3. Discs scan
4. Check of Winsock Layered Service Provider (SPI/LSP)
LSP were checked. Errors weren't found
5. Search for event interceptors that use keyboard/mouse/windows (Keylogger, Trojan DLL)
>>> C:\Program Files\SAMSUNG\MagicKBD\SITKbdHk.DLL --> With high level of probability Keylogger or Trojan DLL was found C:\Program Files\SAMSUNG\MagicKBD\SITKbdHk.DLL>>> Neuronet: file looks like typical event interceptor which uses keyboard/mouse with probability 99.91%
Note: Suspected files DON'T need to be deleted, they should be sent for further analysis (details are in FAQ), because there is big number of useful DDL-interceptors.
6. Search for opened TCP/UDP ports, which are used by harmful programs
There are 319 port descriptions in a base
On this PC there are opened 31 TCP ports and 35 UDP ports
Check was completed, suspicious ports were not found
7. Heuristic check of the system
Check was completed
Number of scanned files: 504, number of extracted from archive: 0, number of harmful programs that were found 0
Scan completed at 05.04.2006 0:01:23
Scan lasted 00:00:55

# References


[1] http://vx.netlux.org/lib/vid00.html
[2] http://vx.netlux.org/lib/?index=PO&lang=EN#start
[3] http://www.fcenter.ru/online.shtml?articles/software/utilities/12214
[4] http://www.symantec.com/avcenter/reference/striker.pdf
[5] http://z-oleg.com/secur/avz_doc/