Other techniques of polymorphism.

ivanlef0u.fr/repo/madchat/vxdevl/vdat/epothpol.htm

Polymorphism is for viruses one of the must. Buz[FS] brings us some valuable ideas for the coding. His paper is very consistent and good written. But there are several ommited things that we should mention.

Interesing idea of complicating scanning, first it was shown in real life by virus IDEA - because it uses cryptographic algorythm named Idea to encrypt its body. It pushed time of emulation of such a decryptor to the limits so antivirus will abort its emulation on time-out. Because even virus itself doesn't know decryptor key and it tests all combinations to find it out. It tooks for example a second, but for emulator in antivirus it will took tens or even hundred of seconds - which is not acceptable of course. But you should keep in mind that it is enought for antivirus to detect decryptor (or even less specific things) to signalise a virus, and there is no real need of such brute-force key finding for antivirus. But if this algorythm is polymorphics enought and antivirus can't detect any scheme in it, this will really work pretty well.
You should also keep in mind to use a good cryptohraphic algorythm (not a simple xor) becase otherwise antivirus can perform a cryptographic analysis faster than is your key-finding routine.

Opcodes variability

You can hear in these days: this poly engine uses fpu instructions, another poly engine uses pentium opcodes, and other one using mmx opcodes. All this sounds good, but is not compatible at all. For example older Cyrix or AMD cpus doesn't have MMX at all. And there are pentiums without mmx and even 486s as well. On those your virus will hang - ant that is best way of its detection by lame users.
Yes it is good to use many specific opcodes, because it will be harder to identify and harder to trace. However you should not use opcodes that are incompatible. How to solve this? Well, my suggestion is to have some extra opcodes enabled by a special flags. Because PEs are basical i386 compatible, you should stay at this level for regular files. But when a virus is going to infect system files to establish itself a home on new computer (like installing to DLLs or VXDs), you can use as many opcodes as current machine supports. Because there is no chance (or very little) that these files will leave current computer. But for transfering virus, you don't know what processor target machine have and you should stay as compatible as original file you are infecting is (to check a CPU flag in PE header). For these reasons, you can read another our article about opcodes.

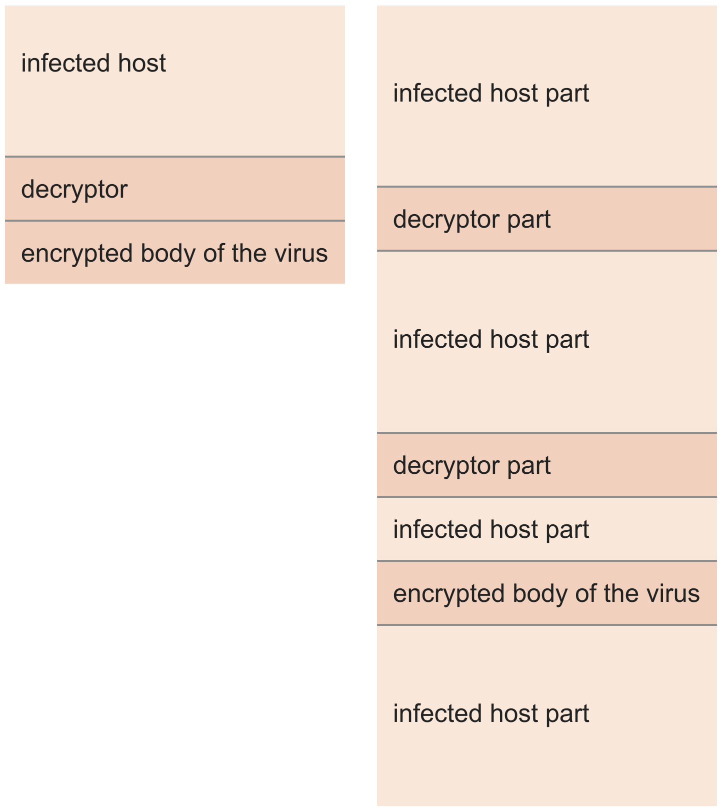
Entry-point hiding

Now, we have to break most common definition of polymorphism associated worldwide. Everyone understoods that polymorphics virus means virus stored in file with fixed body, with generated decryptor to decode fixed body. It is used to prevent easy detection of body instead of it, a generated decryptor must be analysed and detected. But it is not right. This is only way how everyone knows it, however there are also other techniques that breakes this rule. Entry-point hiding, firstly very successfuly demonstrated in Dark Avenger's (in fact inventor of now known polymorphism) piece of code called Commander Bomber. Commander bomber leaves its body completly visible (what a lucky for avers), but you dont know where it actually is. It infects only com files, so whole file can be scanned of course to detect it (a weak point of this virus), but in general you don't know where the body is: there are several fragments of code, place anywhere in host file, that are connected with jumps, contitional jumps and call/rets as well. As it is generated (as well as for classic polymorphical engines) it is hard to identify if fragment of code belongs to Commander Bomber or not. Commander Bomber uses excelent code generator but imho Darkie wanted not to have it encrypted to simplyfo work of avers. No matter now.
This technology is hard to scan, because antiviruses are not loading a whole file (imagine running this on 1mb PE), and simply can't reach body by following all code fragments.

Distributed decryptor

This is some kind og combination idea of hiding entry-point mentioned above with decryption routine. In normal poly engine the situation is similar to figure 1 while distributed poly decryptor look like on figure 2

fig. 1	fig. 2



Prelude to the topic distributed decryptor has been written by Bulgarian programmer known as Dark Avenger in his Commander Bomber virus (already mentioned). The first real (as far as I know) but weak implementation of distributed decryptor can be seen in Vyvojar's One_Half virus with its decryptor divided in 10 parts. However, it was really easy and we should not call it really polymorphic as encryption schema was pretty visible even for stupids. But even as it was so simple, it complicates life to avers really good. May be you remember.

And what would be the perfect distributed decryptor? Imagine decryptor spread all across the host file, with no specific locations, emulated of cos, code fragments linked together with conditional and unconditional jumps, calls, loops combining linear and cyclyc structures, time-out attacks, armouring and anti-debug code. Easy to say, harder to code but why not to try it? A demonstration of this is for example Vyvojar's EMM3 (Explosion Mutation Machine 3).

We can't stop the way of polymorphism on encryptor level. Another level of polymorphism - permutated (we can call it polymorphical, if you want) virus body itself. It is the easier degree of having whole virus in different way every time. It was firstly demonstrated in Ender's TMC:Level_42 that we have also available in this issue (or bugfixed version TMC:Level_6x9 - if you know Hitch Hiker's guide to the galaxy). TMC stands for Tini Mutation Compiler, which is not a good name in fact - because it is a Mutation Linker instead. It is able to place its own code fragments to different locations breaking them at instruction level, connecting these fragments with original conditional jumps or generated jumps, and link all the jumps

and memory references to correct offsets.
We can define code permutating as changing memory position but keeping code-flow of virus code itself. This is rather enought to cause big problems to scanners, as they have to catch all the samples. By choosing any string avir might fail as virus can be breaked within a string and will not be detected. For doing this, virus have to have its own code stored in some form capable for permuattion (that have linking information), or to have some rules how to permutate already running code (and some way to keep linking information as well).

Can virus body be really different for every instance at the instruction level? Well, nowadays there isn't any virus doing this. However I think it is possible. Because there are many ways how to program same subroutine (that even uses same algorythm) and can be completly different at binary and instruction level. It is most probably needed to have some pre-compiled form that will be assembled each time, instead of using its own code as an template (it might be possible, but even much harder to implement). These ideas are more detaily written in Navrhar's article discussing this called ASM vs. HLL.