

TECHNICAL FEATURE

The New 32-bit Medusa

Péter Ször
SARC, USA

I remember the first time I was faced with MtE (the Dark Avenger Mutation Engine). Initial research showed that most anti-virus products were unable to detect 100% of the MtE-based viruses. Product tests carried out by Vesselin Bontchev at VTC showed that most scanners missed a certain percentage (occasionally as high as 10%) of infected files. If infected files were replaced from backups, sooner or later this initial 10% miss could build up to 100% on a particular system. Everything could be infected but the scanner would not be able to detect a single infection!

As virus writers developed polymorphic engines, scanners became stronger in their ability to defend against them. A virus scanner which used a code emulator to detect viruses looked like it was on steroids compared to those without emulator (virtual machine)-based scanning engines.

Nowadays, most polymorphic viruses are considered boring. Even though they can be extremely hard to detect, most of today's products are able to deal with them relatively easily. These are the scanners that survived the DOS polymorphic days; for some others DOS polymorphic viruses signified the 'end of days'. Other scanners died with the macro virus problem. In my opinion, for most products the next challenge is 32-bit metamorphism.

32-bit Encrypted Viruses

Virus writers have always tried to implement virus code evolution from the very early days. One of the easiest ways to hide the functionality of virus code is encryption. Among the first DOS viruses that implemented encryption was Cascade, which starts with a constant decryptor followed by the encrypted virus body.

This simple 'code evolution' method appeared in 32-bit *Windows* viruses very early too. The viruses Win95/Mad and Win95/Zombie use exactly the same technique as Cascade, the only difference being the 32-bit implementation. The detection of such viruses is possible without the trial of having to decrypt the actual virus body; a pattern based on the decryptor is unique enough to identify these viruses. Obviously, such detection is not exact. However, the repair code can decrypt the encrypted virus body and deal with minor variants easily.

32-bit Oligomorphic Viruses

Unlike encrypted viruses, oligomorphic viruses change their decryptors in new generations. Win95/Memorial had the ability to build 96 different decryptors for itself. Thus,

detection of this virus based on the decryptor's code, while possible, was not a practical solution. Most AV products tried to deal with Memorial by dynamic decryption of the encrypted code instead. So detection is still based on the decrypted virus body's constant code.

32-bit Polymorphic Viruses

Win95/Marburg and Win95/HPS were the first viruses to use real 32-bit polymorphic engines. Polymorphic viruses can create an endless number of new decryptors that use different encryption methods to encrypt the constant part of the virus body. Some polymorphic viruses, such as Win32/Coke, use multiple layers of encryption.

Some of the newer polymorphic engines generate a decryptor based on a random decryption algorithm (RDA). Such a decryptor implements a brute force attack against its constant but variable encrypted virus body.

At that time, most scanners already had a code emulator capable of emulating 32-bit executables. Some virus researchers only implemented dynamic decryption to deal with such viruses. That worked in the same way as before because the virus body was still constant under encryption. Next, virus writers used a combination of entry point-obscuring techniques along with 32-bit polymorphism to make the scanners' job even more difficult. In addition, they tried implementing anti-emulation techniques to challenge code emulators.

32-bit Metamorphic Viruses

Virus writers waste weeks or even months creating a new polymorphic virus that is unlikely to get into the wild due to bugs. However, a researcher might deal with the detection of such a virus in a few minutes or at most a few days.

Obviously, virus writers try to implement various new code evolution techniques to make the researcher's job more difficult. Win32/Apparition is the first known 32-bit virus that does not use polymorphic decryptors to evolve itself in new generations. Rather the virus carries its source and drops it whenever it can find a compiler installed on the machine. It inserts and removes junk code to its source and recompiles itself. Thus, a new generation of the virus looks completely different. It is fortunate that Apparition has not become a major problem. However, such a method would be more dangerous if implemented in a Win32 worm.

The Win32/Apparition virus' technique is not surprising. It is much simpler to evolve the code in source format instead of binary. Not surprisingly, many macro and script viruses use a junk insertion and removal technique to evolve themselves in new generations. Igor Muttik explained metamorphism very concisely: 'Metamorphics are body-

polymorphics.’ Metamorphic viruses have neither a decryptor, nor a constant virus body. They do not use a constant data area filled with string constants, but have one single code body that carries data as code.

Although there are some DOS metamorphic viruses, such as ACG, they have not become a significant problem for users. In a few short months the number of metamorphic 32-bit Windows viruses will probably exceed that of metamorphic DOS viruses. The only difference between the two is their potential. The networked enterprise gives metamorphic binary worms the opportunity to cause major problems. And as a result we will not be able to close our eyes to them and say ‘we do not need to handle them since they are not causing problems to our users.’ They will.

In December 1998, the virus writer Vecna created the Win95/Regswap virus. Regswap implemented metamorphism via register usage exchange. Any part of the virus body will use different registers but the same code. Obviously this is not all that complex. Below is a sample code piece selected from two different generations of Regswap.

```

5A          pop    edx
BF04000000  mov    edi,0004h
8BF5          mov    esi,ebp
B80C000000  mov    eax,000Ch
81C288000000  add   edx,0088h
8B1A          mov    ebx,[edx]
899C8618110000  mov   [esi+eax*4+00001118],ebx

58          pop    eax
BB04000000  mov    ebx,0004h
8BD5          mov    edx,ebp
BF0C000000  mov    edi,000Ch
81C088000000  add   eax,0088h
8B30          mov    esi,[eax]
89B4BA18110000  mov   [edx+edi*4+00001118],esi

```

The bold areas show the common areas of the two code generations. Thus, a wildcard string could be useful in detecting this virus. Moreover, support for half-byte wildcards such as 5? B? (as described by Frans Veldman) could lead to even more accurate detection.

However, depending on the actual capability of the scanning engine, such a virus might need algorithmic detection due to the missing support of wildcard search strings. If algorithmic detection is not supported as a single database update, the product update might not come out for several weeks or months for all platforms!

Other virus writers have tried to recreate older permutation techniques. The Win32/Ghost virus can reorder its subroutines like the BadBoy family of DOS viruses. The order of the subroutines will be different from generation to generation, and this leads to $n!$ different virus generations where n is the number of subroutines. BadBoy had 8 subroutines leading to $(8! = 40,320)$ different generations. Discovered in May 2000, Win32/Ghost virus had 10 functions $(10! = 3,628,800)$ combinations. However, both of them can be detected with search strings. Still, some scanners need to deal with this kind of virus algorithmically.

Two different variants of Win95/Zmorph appeared in January of 2000. The virus’ polymorphic engine implements a build-and-execute code evolution. Zmorph rebuilds itself on the stack with push instructions. Blocks of code decrypt the virus from instruction to instruction and push them to the stack. The build routine is already metamorphic. The engine supports jump instructions and removal between any build code instructions. Regardless, code emulators can be used to deal with the virus easily. The virus’ constant code area provides identification since the virus body is decrypted on the stack.

The Win32/Evol virus – which implements a metamorphic engine – appeared in early July. Evol is capable of running on any major Win32 platform. Below is a sample code piece mutated to a new form in a new generation of the same virus. Even the constant-looking double word values can change in the pattern in newer generations, since the virus can calculate them (e.g. Magic=A+B). Therefore, any wildcard strings based on them will not detect anything above the third generation of the virus. Evol’s engine inserts garbage in between core instructions. Here is an early generation:

```

C7060F000055  mov [esi],5500000Fh
C746048BEC5151  mov [esi+0004],5151EC8Bh

```

and one of its later generations:

```

BF0F000055          mov edi,5500000Fh
893E          mov [esi],edi
5F          pop edi
52          push edx
B640          mov dh,40
BA8BEC5151          mov edx,5151EC8Bh
53          push ebx
8BDA          mov ebx,edx
895E04          mov [esi+0004],ebx

```

Members of the Win95/Zperm family appeared in June and September 2000. This virus employs the same infection method as the PLY DOS virus. It inserts jump instructions into its code. The jumps will be inserted to point to a new instruction of the virus. The virus body is built in a 64 KB buffer that is originally filled with zeros.

Zperm will not use decryption. In fact, it will not regenerate a constant virus body anywhere. Instead, it creates new mutations by the removal and addition of jump instructions as well as garbage instructions. Thus, there is no way to detect the virus with search strings in either files or memory. Most polymorphic viruses decrypt themselves to a single constant virus body in memory. However, metamorphic viruses do not. Therefore, the detection of the virus code in memory needs to be algorithmic. Figure 3 explains the code structure of Zperm-like viruses.

```

instruction 2.
JMP instruction 3.
instruction 1. < Entry point>
JMP instruction 2.
instruction 3.
JMP instruction n.

```

Sometimes the virus replaces certain instructions with other equivalent ones. For example, the instruction 'xor eax, eax' (which sets the eax register to zero) will be replaced by 'sub eax, eax' which also zeros the content of the eax register. The opcode of these two instructions will be different.

The core instruction set has the very same execution order; however, the jumps are inserted at random places. The B variant of the virus also uses garbage instruction insertion and removal such as 'nop' (the 'do nothing' instruction.). It is easy to see that the number of generations can be at least 'n' where 'n' is the number of core set instructions in the virus body.

Zperm introduced the RPME (Real Permutating Engine). RPME is available for other virus writers to create new metamorphic viruses. In October 2000, two virus writers created a new metamorphic virus, Win95/Bistro, based on the sources of Zperm and the RPME engine. To complicate matters, this virus uses a random code block insertion engine. A randomly activated routine builds a 'do nothing' code block at the entry point of the virus body prior to any active virus instructions. When executed, the code block can generate millions of iterations.

Win95/Bistro not only mutates itself in new generations. It also mutates the code of its host by a randomly executed code morphing routine. The virus might generate new worms and viruses this way. Moreover, the repair of the virus cannot be done perfectly because the entry point code area of the application can differ. The code sequence at the entry point of the host application will be mutated for 480 bytes. Figure 4 shows an original and a permutated code sequence of a possible entry point code.

Original entry point code:

```

55      push  ebp
8BEC   mov   ebp, esp
8B7608  mov   esi, dword ptr [ebp + 08]
85F6   test  esi, esi
743B   je    401045
8B7E0C  mov   edi, dword ptr [ebp + 0c]
09FF   or    edi, edi
7434   je    401045
31D2   xor   edx, edx

```

Permutated entry point code:

```

55      push  ebp
54      push  esp
5D      pop   ebp
8B7608  mov   esi, dword ptr [ebp + 08]
09F6   or    esi, esi
743B   je    401045
8B7E0C  mov   edi, dword ptr [ebp + 0c]
85FF   test  edi, edi
7434   je    401045
28D2   sub   edx, edx

```

Thus an instruction such as 'test esi, esi' can be replaced by 'or esi, esi', its equivalent format. A 'push ebp, mov ebp, esp' sequence (very common in high level language applications) can be permutated to 'push ebp, push esp, pop

ebp'. Obviously it would be more complicated to replace the code with different opcode sizes but it would be possible to shorten longer forms of some of the complex instructions and include 'do nothing' code as a filler.

This is a major problem for all AV scanners. Heuristic scanners typically cannot deal with high level language written worms yet. Obviously some of these worms could easily be morphed to a new format. In my VB2000 conference paper I already introduced the problem of new virus variants being generated accidentally as a result of Portable Executable file repair. While it is unfortunate that such mutations can appear, it is feasible to deal with the problem. On the other hand, code permutations of worms and viruses, as performed by Win95/Bistro, will be much more difficult to deal with.

If a virus or a 32-bit worm capable of implementing a similar morphing technique should appear, the problem could be major. New mutations of old viruses and worms would be morphed endlessly and a virtually endless number of not-yet-detectable viruses and worms would appear without any human intervention, leading to the ultimate virus generator.

At the end of 1999 the Win32/Smorph Trojan was developed. It implements a semi-metamorphic technique to install a backdoor to the system. The standalone executable is completely regenerated during the installation of the Trojan. Its PE header will also be new and will include new section names and section sizes.

The actual code at the entry point is metamorphically generated. This code will allocate memory, then decrypt its own resource that contains a set of other executables. The Trojan uses API calls to its own import address table. The import table is filled with a lot of non-essential API imports as well as some essential ones. Thus, everything in the standalone Trojan code will be different in new generations.

Conclusion

It is only a matter of time until we see in-the-wild Win32 worms using metamorphic engines. Unfortunately, metamorphic viruses such as Win95/Bistro often have a random replication mechanism. Since their code structure is much more obfuscated, they are more difficult to analyse than polymorphic viruses. Their random infection and spreading mechanism will make the job of automated analysers and advanced behaviour-blocking systems more challenging.

We need to support detection of such viruses regardless of their complexity. It seems that scanning technology has to go through a new evolution! It is clear that by the time meta-morphism in viruses becomes complex, scanning technology alone will be inefficient as a primary anti-virus defence solution. It is going to be extremely difficult to deal with the rising number of potential false positives. Therefore, we must start to develop new systems and defences to reduce the inevitable overload in the future.