Guide to improving Polymorphic Engines

ivanlef0u.fr/repo/madchat/vxdevl/vdat/tumisc17.htm

Guide to improving Polymorphic Engines
by Rogue Warrior

Table of contents :-

===

Introduction :-

This is a guide for those who already know how to make an engine but cannot work out why their viruses are still detectable.

The single purpose of polymorphic viruses is to avoid detection - at the heart of the polymorphic virus is the engine. It can usually take from 30-80% of the virus code size so is a very important component of the virus to have working properly.

This guide will tell you how polymorphic detectors work in order to help you design/make a better engine to defeat scanners.

Making a good engine takes a good amount of time. If you don't make it correctly you might as well leave it out completely because it's main purpose (avoiding detection) will not work!.

===

Levels of Polymorphism :-

Polymorphism covers has many levels of skill.

According to Vesselin Bontchev (AV) these are :-

1. Fixed set of constant decryptors (a.k.a Oligomorphic).
2. Variable instructions for single instruction.
3. Garbage code insertion.
4. Instruction swapping.
5. 2+3+4

----

6. Permuting

#6 is not considered higher than #5 it's simply considered a different classing.

I think there is now a 7th class. Highly advanced polymorphism which is designed to be better than #5. These ones have the following attributes:

* Heuristic counter-measures

    * Goat counter-measures
    * Emulator counter-measures

  All these attributes are not part of the virus but instead part
  of the _polymorphic code produced by the virus_.

==

Polymorphic Virus Detection Methods :-

  There are many methods for detecting polymorphic viruses here are some
  popular methods:

- Scan Strings
- Variable Scan Strings
- Cryptanalysis
- Generic Decryptor
- Heuristics

```
  Scan Strings ::-
       Works by searching for a pattern of bytes in FIXED positions and a
       FIXED sequence.

       e.g.,

       scan string: aa ?? bb ?? cc
       virus text:  aa xx bb xx cc

  Variable Scan Strings ::-
       Work by searching for a pattern of bytes in VARIABLE positions
       but in a FIXED sequence.

       e.g.,

       scan string: aa * bb * cc
       virus text:  1. aa xx xx bb xx xx xx xx cc
                    2. aa bb xx xx xx cc
                    3. and so on...

  Cryptanalysis ::-
       Works by finding part of the VIRUS BODY and then performing some
       very basic cryptanalysis on it and then decrypting it (if possible).

       This method according to many AV is not used anymore (due to the
       effectiveness of Generic Decryptor) but I will tell you how to
       defeat it anyway just to be sure ;-) -- its not hard to defeat.

  Generic Decryptor (a.k.a. Emulation) ::-
       Works by emulating instructions in the polymorphic decryptor in order
       to make the virus decrypt itself and then it detects the virus by a
       standard scan string.

  Heuristics ::-
       This has undeservedly been a virus buzz word for a long time.  It
       has been the target of polymorph engine creators to beat the heuristics
```

which shows how little they know of polymorphic detection.

This method involves searching for inconsistencies between the code being analysed and normal everyday code found in programs.

While it is important - it is not THAT important and will not help you stop being detected by anti virus software.

It is important to note that heuristics is not used very much (they do use a bit) in the most popular AV programs (F-PROT, McAfee and AVP) these are the programs you should target. Do not target programs which only hard core virus people use. Most of the hard core AV software could spot a virus anyway. -- in other words: _target the less intelligent software users_

==

Combatting Virus Detection Methods :-

- Anti Scan String methods

This is really easy - avoid the use of code common to every decryptor just because some code isn't in the same position doesn't mean it cannot be scanned though. For example:

```
 xx=garbage code.

 Your Decryptor #1 (as hexidecimal):

  45 34 xx xx xx 54 80 xx xx xx 12 xx xx xx 34 32 xx xx xx 43 xx xx xx xx

 Your Decryptor #2:

  xx xx xx 45 30 xx xx xx xx xx 54 81 xx xx xx xx xx 12 xx xx 34 32 xx 43
```

Looking at this code you can see an obvious pattern it can be scanned using this string:

```
     45 3? * 54 8? * 12 * 34 32 * 43

     Legend:
       ?     - match 1 positions only
       *     - match up to N bytes but low as 0 bytes
```

This will identify this decryptor (not the virus) by looking for code common to each decryptor. So how do you combat it? Well try making sure that you always have at LEAST 1 alternative to every instruction your engine can generate.

NOTE: Make enough alternatives that it makes multiple variable scan strings not an option to AV!

Cryptanalysis :-

This is very easy to defeat - simple add multiple encryption operations for example:

A loop using a single XOR with byte/word is very easy to cryptanalyse but a loop using XOR b/w, ADD b/w, SUB b/w, ROL b/w in one loop is VERY hard to cryptanalyse.

The only problem with this is applying the encryptions in reverse order to that of your virus decryptor so that when the virus decryptor is run it will do it in the correct ordering.

There is an easy way to do this! -- There isn't really I was just joking there is no easy way =)

You can leave bit out anyway because AV's are using all using Generic Decryption as far as I know.

Generic Decryption :-

This method is very popular amongst AV and requires the cooperation of the virus to work. If a virus can detect it is being emulated and then throw an emulator off by some method then it will defeat this method.

Products known to be using this technique are: F-PROT AVP TBAV DSAV (and I would guess McAfee?).

How does generic decryption work? well the AV products each have in them a little Intel software CPU emulator which does not allow instructions to actually execute but simulates them enough in certain controlled ways in order to make the virus decrypt itself in a safe environment - this way all they need is a scan string for a very complex polymorphic virus!

These controlled conditions avoid endless loops and other similar bugs in normal programs from making the emulator hang. I experimented and found that making a 10 KB decryptor on a virus will dramatically slow down scanning in DSAV and AVP because the emulator is actually simulating the code. I made 10 x 10KB samples and these took over 3 minutes to be examined by DSAV and AVP however each of these took only milliseconds to run normally.

This shows that the emulators in DSAV and AVP are really quite good and don't give up easily when trying to decrypt a virus (I used the /ANALYSE option on DSAV). F-PROT and TBSCAN did not emulate these samples correctly even with maximum heuristics enabled or if they did they must have discovered how to simulate INCREDIBLY quickly (even TBSCAN being written in assembly language cannot run them that fast).

So how do we stop this emulation taking place? or better put: How do we detect ourselves being emulated and throw the emulator off?

e.g., Imagine we know that the PSP contains a certain constant value at ALL times - but we also know the emulator doesn't emulate the PSP. With this knowledge we can construct some code in our polymorphic DECRYPTOR to detect this and throw the emulator off:

```
        mov      ax,[0000]
        sub      ax,20CDh
        jz       ok
        mov      ah,4Ch
        int      21
        ok:
```

Note: This code must be in the decryptor because it's goal is to stop decrypting BEFORE we reach the virus body. This code must be generated with the same principles of variability that all other poly code requires - if you don't make this code variable also then you risk having the code used against you to detect the virus!!!

Possible methods to exploit for detecting and terminating emulation:

- Inability of DOS call return values to be predicted by the emulator without actually calling them. i.e., we can make a DOS call in the poly code and check the return values. If they are not consistent with a REAL call to that function then it can be assumed we are being emulated and then take evasive action.

- Inability of emulator to write to ALL of memory. By writing to safe areas in RAM we can test if that area has ACTUALLY been written to by the virus or just emulated. If it has not been written to then we assume emulation in progress.

e.g., long winded version:

```
       cli                       ;disable ints
       cld                       ;set data string copy direction
       push     6000h
       pop      ds               ;any segment which AV and virus don't own.
       push     ds
       pop      es               ;es=ds=6000h
       sub      si,si
       mov      di,0002
       lodsw                     ;save in ax, si=di=2
       xor      ds:[0000],1234h ;write
       mov      cx,0f000h        ;some large amount
L1:    rep      movsw            ;write to memory (a large amount is better)
       cmp      ds:[0000],ax     ;did the AV forget about the write?
       mov      ds:[0000],ax     ;set it back to normal regardless
       jz       not_emulated     ;seems they messed up remembering where we
                                 ;wrote.
       mov      ah,4Ch
       int      21h              ;bye Mr Emulator.

not_emulated:
```

- Inability of emulator to emulate all control structures (PSP, MCB, SFT, etc).

Most emulators can emulate the PSP, MCB and so on but every single structure would take too much memory and processing so trying to exploit this possible weakness is a good idea.

TbClean a program which emulates viruses to disinfect programs only emulates certain small parts of the PSP leaving other parts to be exploited by emulation trapping. In fact one can trick TbClean into converting the virus infected file into an infected Trojan horse program for the person who runs it next.

NOTE: TbClean is good fun for testing your polymorphic decryptors it shows you how the emulator is going to go through your code like a hot knife through butter. Make sure to crack the registration on TbClean so you can use it properly <grin>.

- Limited resources of an emulator.

Remember that many AV programs are built to be fast so by making your virus take a very long time (in AV program terms) to analyse your virus might make it quit thinking that it has encountered an endless loop.

However! running a time consuming decryptor normally takes next to no time. So we can see that resources of time, memory, processing power all contribute to methods for killing off an AV scanner emulator.

*** You must think how to detect and force the emulator to quit ***

==

Stopping analysis of your virus by AV researchers

AV researchers are the ones responsible for making your virus detectable so having some ways to hinder AV researchers doing analysis of your polymorphic virus and engine is always good to throw in.

The most common way to analyse a new polymorphic virus is to generate 1000's of samples of your virus. This involves activating the virus on a test computer and executing 1000's of goat programs.

The goal in generating these 1000's of copies is to get a good sampling of what the engine can generate and then test the detection method against it.

If your virus chooses only to show a certain sample then their detector may work in the Lab but not when it comes to "in the wild" situations.

Of course it is best to not make it obvious to AV that you are trying to do this or they might catch on and alter their methods.

==

Planning your engine features

It's always good idea to have plan of the engine structure.

Many coders spend their time byte-fiddling trying to optimise
their code - this method of planning enables to you block-fiddle
- each of these blocks can be shuffled and optimised meaning
every change for the better is saving you lots of bytes instead
of 1-2 bytes.

NOTE: NEVER place ANY code in a CALL/RET routine unless it
      is used more than once!

A polymorphic engine is very similar to the code generation
phase of a compiler - most compiler writers use the word "emit" [1]
as the word to say they're outputting code. So try to use
the same because it's good to follow this standard when
planning your engine.

[1]: Means "output" or "give off" for those bad at English.

e.g., Here is a very basic model of an engine plan
      (you may want to add more detail than this
       to any plan you make):

```
 Engine:
   EmitDecryptor

 EmitDecryptor:
   repeat EmitGarbage & EmitAntiEmulation, random(5) times
   EmitSetupRegs
   repeat EmitGarbage & EmitAntiEmulation, random(5) times
   MarkLoopStart
   repeat EmitDecryptionCode, random(5) times
   repeat EmitGarbage & EmitAntiEmulation, random(5) times
   EmitEndLoop
   repeat EmitGarbage & EmitAntiEmulation, random(5) times
 End-EmitDecryptor

 EmitGarbage:
   Randomly Select 1 of:
     EmitFakeINT21   - randomly select some int 21 functions
     EmitFakeINT10   - randomly select some int 10 functions
     EmitCMPbmemXX   - cmp byte ptr [xxxx],xx
     EmitCMPwmemXXXX - cmp word ptr [xxxx],cccc
     EmitMOVbmemXX   - mov byte ptr [xxxx],cc
     EmitMOVwmemXXXX - mov word ptr [xxxx],cccc
     EmitMOVbregXX   - mov rb,cc
     EmitMOVwregXXXX - mov rx,cccc
     EmitMOVbregMEM  - mov rb,byte ptr [xxx]
     EmitMOVwregMEM  - mov rw,byte ptr [xxxx]
     EmitCALL        - CALL xxxx/garb/jmp yyyy/garb/xxxx:/garb/ret/yyyy:
     EmitJMP         - JMP xxxx/garb/xxxx:
```

```
End-EmitGarbage

EmitAntiEmulation:
  Randomly Select 1 of:
    EmitFarCALL       - place RETF into mem/CALL yyyy:xxxx
    EmitFarJMP        - place Far JMP into mem/JMP yyyy:xxxx
    EmitWriteAndTest  - write to known RAM mem, test it changes, if not
                        crash
    EmitFakeExit      - set int 21 = virus_cs:virus_return and call
                        ah=4c, int 21
    EmitPSPcheck      - cmp ds:[0000],21CDh/jnz crash: use better check!
                        just an example.
    EmitDOScheck      - dos call/check return value is consistent.
End-EmitAntiEmulation

EmitSetupRegs:
  If Boolean Then
    LoopType = Counter or Pointer
    Select Count Register from [AX,BX,CX,DX,SI,DI]
    Select Pointer Register from [SI,DI,BP]
  Else
    LoopType = Pointer
    Select Pointer Register from [SI,DI,BP]
  End If
End-EmitSetupRegs

MarkLoopStart:
  Save output pointer (usually DI register) to remember loop location.
End-MarkLoopStart

EmitDecryptionCode:
  Randomly select 1 of:
    EmitXORptr
    EmitADDptr
    EmitSUBptr
End-EmitDecryptionCode

EmitEndLoop:
  If LoopType=Counter and Counter=CX and Boolean Then
    EmitLoop
  Else If LoopType=Counter and Boolean Then
    EmitDECJNZ
  Else If LoopType=Counter and Boolean Then
    EmitDECJZJMP
  Else If LoopType=Pointer and Boolean Then
    EmitDECCMPJNZ
  Else If LoopType=Pointer and Boolean Then
    EmitDECCMPJZJMP
  End
End-EmitEndLoop

This is just a simple example of a plan so you can see how to
structure your engine - do not forget these parts:

  - encrypting the virus body in reverse order and
```

```
       reverse operation.

     - adjusting for execution location in memory:
       if the entry point of the virus is not at a zero
       offset then you must adjust all memory references
       and pointers by the relocation amount.

       This part is usually done while emitting.

============================================================================
```

Conclusion

If you are going to go the trouble of making a polymorph engine then do it right and don't waste 1-3Kb of code on an engine which can be generically decrypted.

If you are going to make a good engine remember the following points:

- it must not have fixed bytes in fixed positions.
- it must not have fixed bytes in variable positions.
- it must not be able to be decrypted by generic decryption engines in AV software.
- it helps if the code is heuristically "clean" but it is not the be all and end all of an engine to be this way.
- make sure it is a bitch to analyse by AV.
- make sure it is a bitch to remove if it does get caught.

The final pain in the ass

Some AV are obsessed with EXACT detection - even if they are able to detect your decryptor like they do to many some TPE based viruses - in the end they always want exact detection. So try to make your engine such a hard ass that it might allow detection of the actual "decryptor" part but NOT the virus body - This will be a great annoyance to the AV (even though they may say otherwise).

Remember inexact detection leads to inability to remove the virus and if you virus ever becomes common they will have to answer the customers question "why can't you remove it ?".