# Obfuscation procedure based in Dead Code insertion into Crypter

Cristian Barría
Pontifícia Universidad Católica de Valparaíso
Valparaíso, Chile
cristian.barria@udp.cl

Claudio Cubillos
Pontifícia Universidad Católica de Valparaíso
Valparaíso, Chile
claudio.cubillos@ucv.cl

David Cordero
Universidad Andrés Bello
Santiago, Chile
d.cordero.v@gmail.com

Robinson Osses
Universidad Mayor
Santiago, Chile
robinson.osses@mayor.cl

*Abstract* — ***The* threat that attacks cyberspace is known as malware. In order to infect the technologic devices that are attacked, malware needs to evade the different antivirus systems. To avoid detection, an obfuscation technique must be applied so malware is updated and ready to be performed. No obstant, the technique implementation presents difficulties in terms of its required ability, evasion tests and infection functionality that turn outs to be a problem to keep malware updated. Therefore, a procedure is proposed that allows applying AVFUCKER or DSPLIT techniques. The purpose is to optimize the required technical means, reduce the antivirus analysis and malware functionality check times.**

***Index Terms- malware; obfuscation; modding; antivirus; evasion.***

## I. INTRODUCTION

Cyberspace is a system of systems that is permanently evolving. It transcends space limits and it is composed of interdependent network infrastructures mainly, which include other systems: Internet, telecommunications, electronics, corporate computer systems, servers, processors and controllers; all together producing data and the information as an essential consequence. Cyberspace is not a natural environment, created by human beings and has potential benefits as well as unknown risks [1]. It operates in the electromagnetic spectrum and their physical nodes are in the traditional domains (earth, sea, air and space). When they are interrelated, they have implications associated to the risks that may be passed on each other. These nodes could be managed or operated by private or governmental organizations [2].

These threats coming from high level sophistication malware are meant to damage critical infrastructure, essential services or the command and control servers. In the presence of those attacks, the affected nation Executive Power can, occasionally, arrange the neutralization of those damaging servers (Article 51 of United Nations Charter).

In order to evade the antivirus, the manual update that the malware requires, is without a doubt, one of the most complex challenges that needs to be solved. Methods, techniques, procedures and tools application are necessary to achieve malware obfuscation [3]. This needs to use the encryption method that requires a tool named “Crypter” [4], which allows using obfuscation in any kind of binary state file. Is easy to obtain this tool in security specialized websites, which help us to encipher malware, so it keeps its original functionality and avoids malware detection using reverse engineering.

A crypter is composed of two parts [6]: a) builder, parts that is in charge of enciphering the binary code (input) and then connect it to the stub. This process creates an enciphered malware plus the stub (output); b) Stub, is the most important component of the tool, since its function is to decipher the binary code (malware) generated by the builder, using a key that activates the cryptographic algorithm, executing it directly on the memory, without passing over the hard disk [4], as it is shown on the figure 1. This function is essential, since the antivirus performs the scan on the disk.

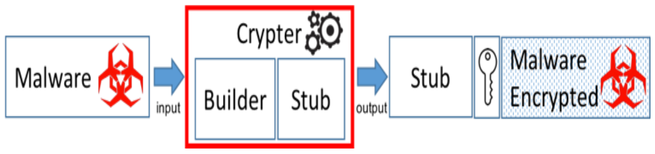


*Figure 1. Crypter components*

Due to the features that this tool has to encipher data and run it on the memory, antivirus analyzes stub to the crypter output. This part is not ciphered, so it is possible to read and therefore to evaluate it with the antivirus engines. In case it is detected as a malicious code, the crypter would be then immediately considered within the same category [7], residing here the problem that is exposed on the present research, according to the encryption method for the manual update.

Other authors also mention similar techniques and tools for the code obfuscation, for example Mohanty [8], who mentions the use of two applications referred as Mistfall [9] and Burneye [10], although he does not precisely indicates how to use them. Bashari [11], states which are the most common obfuscation techniques, such as dead code insertion, variable/register substitution, instructions replacement and permutation; but he does not demonstrate their functionality on malware. Ammann [4], for his part, indicates the crypter use as an infection mean due in this way it is possible to keep the malware original behavior, but he does not include obfuscation techniques. Singh [12] mentions Split and Binding techniques, which allow avoiding antivirus if they come with obfuscation methods, but he does not indicate which ones can be used. In other words, all of these authors tell us what to do but they do not tell us how, so malware keeps undetectable.

This is why, in case of stub detection, it is necessary to perform an obfuscation procedure, so it is possible to apply the crypter over malware again. To obtain an updated malware, and therefore its evasion ability, it is necessary to check the adapted stub that is part of this updated malware. In this way, the research contribution is given by the offer of a procedure to achieve the stub undetection, based on the Dead Code Insertion technique on a Crypter.

## II. MALWARE OBFUSCATION PROCEDURE

When considering the diagram that is exposed on the figure 2 [13], it is observed that malware update starts on its binary state, applying on it a method named “encryption”, that requires a crypter to perform. Then, the crypter output must be proven upon an antivirus (signature based analysis) and if it is not detected, malware would be updated. On the contrary, the “Dead Code Insertion” technique [14] must be performed and also an obfuscation procedure. In this case AvFucker must be used so the technique over the crypter is used again on an iterative way, until antivirus achieves the evasion without modifying its functionality of the initial binary state [13].

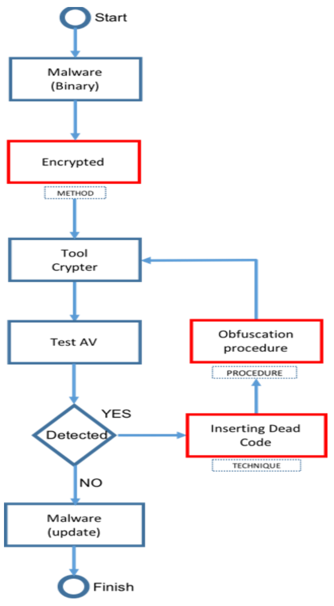


*Figure 2. Manual update malware diagram.*

When the crypter output is contrasted through detection techniques based on signatures [15] and is detected as malware, the procedure is applied as a whole. That means, the stub, the key and the enciphered malware are included, although only the stub area is detected. Such procedure consists on the malware copies creation (crypter output) which quantity is given by the file size, according to the following equation:

$$Nc = (St + K + Mc)$$

Where:
Nc= Copies number.
St= Stub bytes size.
K = Key bytes size.
Mc= Enciphered malware bytes size.

Each of these copies is created for the Dead Code Insertion (00, 90), from byte to byte, from the initial to the final offset, correlatively [16], as it is observed on the figure 3.

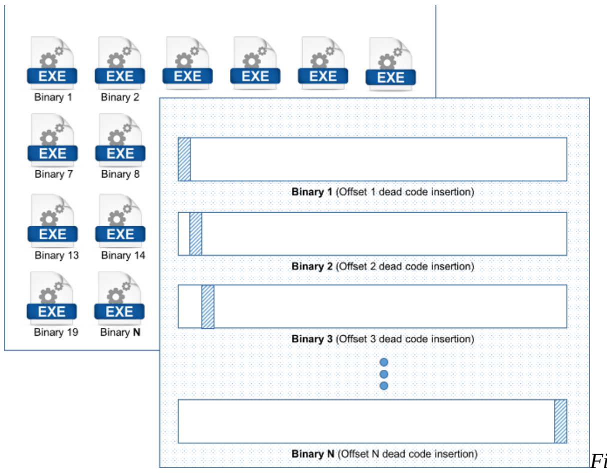

EXE
Binary 1
Binary 2
Binary 7
Binary 8
Binary 13
Binary 14
Binary 19
Binary N
Binary 1 (Offset 1 dead code insertion)
Binary 2 (Offset 2 dead code insertion)
Binary 3 (Offset 3 dead code insertion)
Binary N (Offset N dead code insertion)

*Figure 3. Malware copies generation*

Once all the copies are made, they are analyzed by the antivirus as the figure 4 shows. In this way, only files detected as non malicious are obtained and the rest is removed when matching with the antivirus registered signature. Then, a group of files is obtained that is able to avoid antivirus but its hexadecimal code has been modified, so it is necessary to verify the functionality is not affected.

What was previously presented is not viable if a file of 1 Mb output is used as an example:

$Nc = (St + K + Mc) = 1.048.576$ copies
$Nc^2$ = Space for the copies on the Hard disk
$Nc^2 = (1.048.576)^2$ bytes = 1 Terabyte.

The previous mentioned implies the following problems: a) capacity, since the number of files created could be higher than the information terabytes, affecting directly the hardware capacity, which turns out also in equipment investment; b) test, in function of the time that the antivirus uses to perform the analysis on each one of the copies; c) functionality, revision that must be made on a individual way to the group of undetected files by the antivirus and then prove that the malware functionality is not affected.

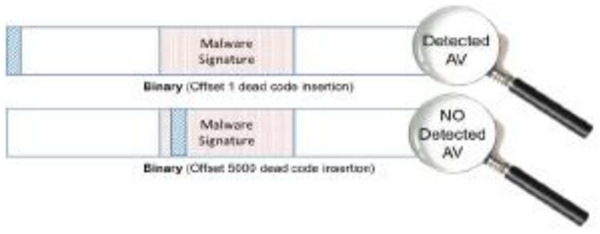

Malware Signature
Detected AV
Binary (Offset 1 dead code insertion)
Malware Signature
NO Detected AV
Binary (Offset 5000 dead code insertion)

*Figure 4. Antivirus analysis.*

This is how the procedure is seeking to be optimized, increasing the Dead Code Insertion. If the 1 Mb output file is used, it is possible to create copies every 1000 bytes (offset with the 0,1 % of the total file size) and the proper analysis is made, considering only the portion of 1000 bytes that gets undetected. Then, copies are created every 100 bytes over it, repeating this procedure until getting to the portion of 1 byte, as the figure 5 shows [13], taking into account that the ranges are not necessarily in a consecutive way. Thus, hardware use increases and the created files number decreases, so the antivirus testing time also decreases.

However, the problem of malware operation test persists because a controlled environment is needed that allows proving functions typical of this kind of malicious code, such as the persistence, connectivity or spreading among others. This takes a lot of time and a great number of resources for the modder [13]. This way, a tool is used and it is called: “annotator” [17], which permits to verify the functions described above, by replacing input malware on the crypter, as the figure 6 shows. Besides, another characteristic is its tiny size, allowing optimization of the creation of copies even more and reducing the antivirus analysis and detection time.

## III. VALIDATION TESTS OF AVFUCKER PROCEDURE OPTIMIZATION PROPOSAL

A flow chart is proposed from the theoretical analysis that permits optimization of the AVFUCKER procedure, as the figure 7 shows.

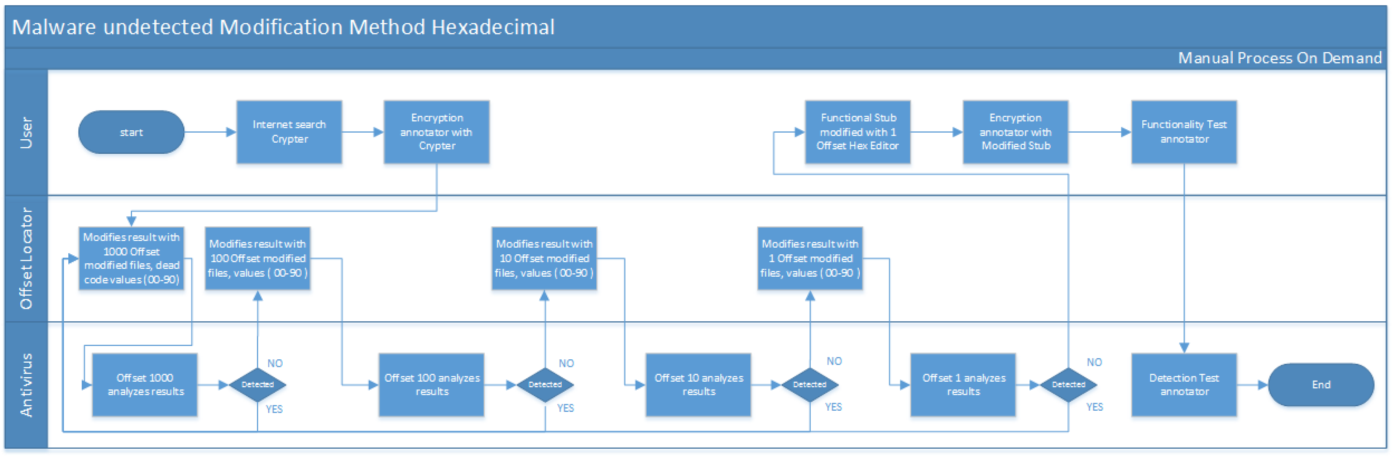
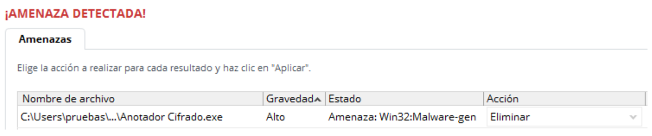

Malware undetected Modification Method Hexadecimal
Manual Process On Demand
User
Offset Locator
Antivirus
start
Internet search Crypter
Encryption annotator with Crypter
Functional Stub modified with 1 Offset Hex Editor
Encryption annotator with Modified Stub
Functionality Test annotator
Modifies result with 1000 Offset modified files, dead code values (00-90)
Modifies result with 100 Offset modified files, values ( 00-90 )
Modifies result with 10 Offset modified files, values ( 00-90 )
Modifies result with 1 Offset modified files, values ( 00-90 )
Offset 1000 analyzes results
Detected
NO
YES
Offset 100 analyzes results
Offset 10 analyzes results
Offset 1 analyzes results
Detection Test annotator
End

*Figure 7 Proposal AVFUCKER optimized process flow chart.*

This is going to be validated empirically, as exposed bellow:

### A. Annotator obtainment

A annotator is obtained from Information Security specialized websites, as “BOLITA.EXE”[18], applying then a crypter that creates an output composed by a stub, a key and an encrypted annotator when being subject to an antivirus is detected, as it is exposed by the image 1.



¡AMENAZA DETECTADA!

Amenazas

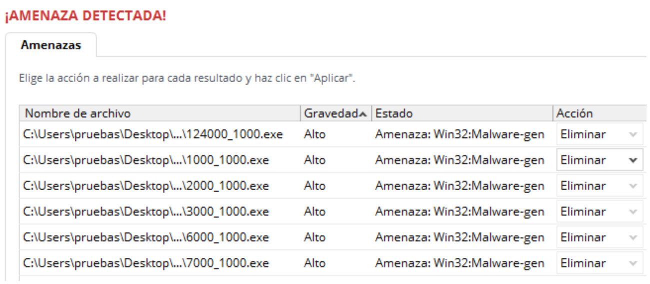
Elige la acción a realizar para cada resultado y haz clic en "Aplicar".

| Nombre de archivo | Gravedad | Estado | Acción |
|---|---|---|---|
| C:\Users\pruebas\...\Anotador Cifrado.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |

*Image 1. Annotator malware detection.*

### B. Procedure application

Being detected by the antivirus, in this case “Avast Antivirus” [19], the procedure application is performed. It is possible to use a tool like “Offset Locator 2.6”[20] for it, considering 1.000 bytes ranges, and it is analyzed again so the group of undetected files is obtained, which for this case is given for offset 4.001 and 5.999 ranges, as is exposed on image 2.



¡AMENAZA DETECTADA!

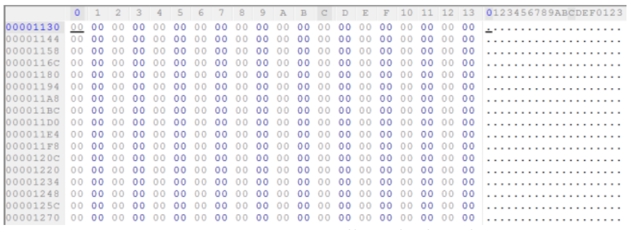
Amenazas

Elige la acción a realizar para cada resultado y haz clic en "Aplicar".

| Nombre de archivo | Gravedad | Estado | Acción |
|---|---|---|---|
| C:\Users\pruebas\Desktop\...\124000_1000.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\1000_1000.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\2000_1000.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\3000_1000.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\6000_1000.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\7000_1000.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |

*Image 2. Rank of 1.000 byte of detected offset.*

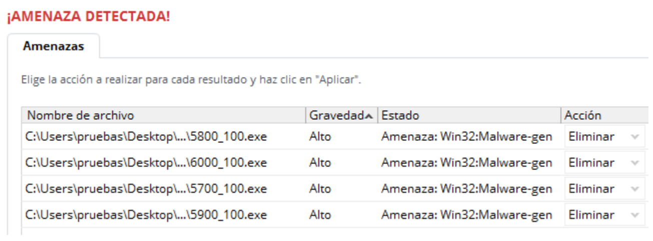
As we can see on the image 3, from the file that was not detected as malware (4.001- 5.000 range), Dead Code Insertion is evident (00) in all its offsets, for which the tool Workshop Hex [21] can be used, assuming that the signature that the antivirus recognizes is found in this sector of the base malware.



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10 11 12 13  0123456789ABCDEF0123
00001130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001144  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001158  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
0000116C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001194  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
000011A8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
000011BC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
000011D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
000011E4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
000011F8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
0000120C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001234  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001248  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
0000125C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
```

*Image 3. Dead code insertion in offset of a file of 1.000 bytes.*

Later, ranges between offset 4.001 y 5.999 are delimited, generating 100 bytes files on this sector to be analyzed again and then obtain the group of undetected files, with range 4.001 y 5.799, as the image 4 shows.



¡AMENAZA DETECTADA!

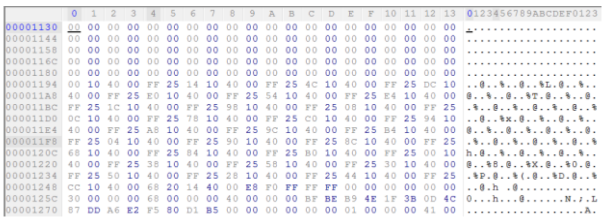
Amenazas

Elige la acción a realizar para cada resultado y haz clic en "Aplicar".

| Nombre de archivo | Gravedad | Estado | Acción |
|---|---|---|---|
| C:\Users\pruebas\Desktop\...\5800_100.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\6000_100.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\5700_100.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\5900_100.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |

*Image 4. 100 byte range of detected offsets.*

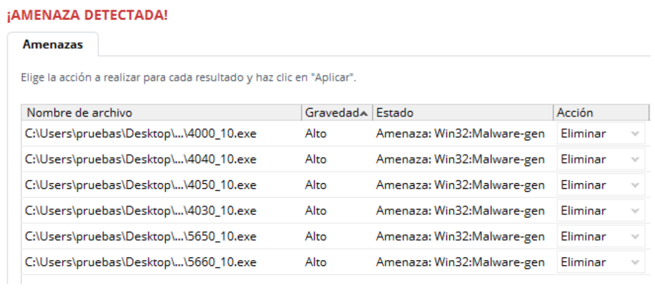
The image 5 exposes the file that was not detected as malware corresponding to the range 4.001 y 5.799, demonstrating the Dead Code Insertion (00) in all its offsets, assuming that the signature that the antivirus recognizes is found in this sector of the base malware.



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10 11 12 13  0123456789ABCDEF0123
00001130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001144  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001158  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
0000116C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00001194  00 10 40 00 FF 25 14 10 40 00 FF 25 4C 10 40 00 FF 25 DC 10  ..@..%..@..%L.@..%..
000011A8  40 00 FF 25 E0 10 40 00 FF 25 54 10 40 00 FF 25 E4 10 40 00  @..%..@..%T.@..%..@.
000011BC  FF 25 1C 10 40 00 FF 25 98 10 40 00 FF 25 08 10 40 00 FF 25  .%..@..%..@..%..@..%
000011D0  0C 10 40 00 FF 25 78 10 40 00 FF 25 C0 10 40 00 FF 25 94 10  ..@..%x.@..%..@..%..
000011E4  40 00 FF 25 A8 10 40 00 FF 25 9C 10 40 00 FF 25 B4 10 40 00  @..%..@..%..@..%..@.
000011F8  FF 25 04 10 40 00 FF 25 90 10 40 00 FF 25 8C 10 40 00 FF 25  .%..@..%..@..%..@..%
0000120C  68 10 40 00 FF 25 84 10 40 00 FF 25 B0 10 40 00 FF 25 00 10  h.@..%..@..%..@..%..
00001220  40 00 FF 25 38 10 40 00 FF 25 58 10 40 00 FF 25 30 10 40 00  @..%8.@..%X.@..%0.@.
00001234  FF 25 50 10 40 00 FF 25 28 10 40 00 FF 25 44 10 40 00 FF 25  .%P.@..%(.@..%D.@..%
00001248  CC 10 40 00 68 20 14 40 00 E8 F0 FF FF FF 00 00 00 00 00 00  ..@.h .@............
0000125C  30 00 00 00 68 00 00 00 40 00 00 00 BF BE B9 4E 1F 3B 0D 4C  0...h...@......N.;.L
00001270  87 DD A6 E2 F5 80 D1 B5 00 00 00 00 00 00 01 00 00 00 41 00  ..................A.
```

*Image 5. Dead Code insertion in offset of a file of 100 bytes.*

Just like the above mentioned items, 10 bytes files on this sector are generated on the offsets from ranges 4.001 and 5.799. Then they are analyzed by the antivirus again to obtain the undetected files, producing in this case several not consecutive ranges between 4.061- 5.649 y 4.011-4.029, as the image 6 exposes.



¡AMENAZA DETECTADA!

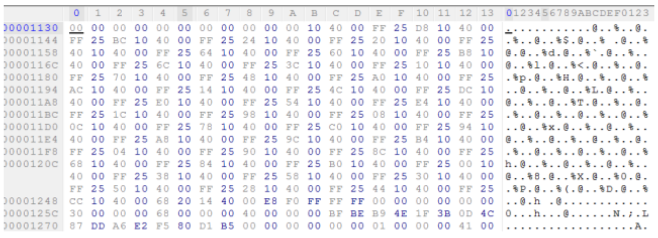
Amenazas

Elige la acción a realizar para cada resultado y haz clic en "Aplicar".

| Nombre de archivo | Gravedad | Estado | Acción |
|---|---|---|---|
| C:\Users\pruebas\Desktop\...\4000_10.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\4040_10.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\4050_10.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\4030_10.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\5650_10.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\5660_10.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |

*Image 6. Range of 10 byte of detected offset.*

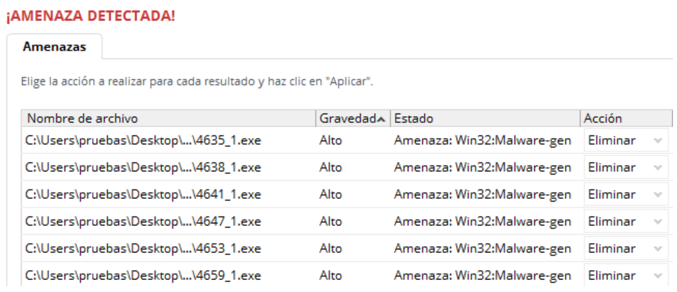
On the image 7 of the file that was not detected as malware between the ranges 4.061- 5.649 and 4.011-4.029, the Dead Code Insertion is evident (00) in all its offsets, assuming that the signature that the antivirus recognizes is found in this sector of the base malware.



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10 11 12 13  0123456789ABCDEF0123
0001130  00 00 00 00 00 00 00 00 00 00 00 10 40 00 FF 25 D8 10 40 00  .............@..%..@.
0001144  FF 25 BC 10 40 00 FF 25 24 10 40 00 FF 25 20 10 40 00 FF 25  .%..@..%$.@..% .@..%
0001158  40 10 40 00 FF 25 64 10 40 00 FF 25 60 10 40 00 FF 25 B8 10  @.@..%d.@..%`.@..%..
000116C  40 00 FF 25 6C 10 40 00 FF 25 3C 10 40 00 FF 25 10 10 40 00  @..%l.@..%<.@..%..@.
0001180  FF 25 70 10 40 00 FF 25 48 10 40 00 FF 25 A0 10 40 00 FF 25  .%p.@..%H.@..%..@..%
0001194  AC 10 40 00 FF 25 14 10 40 00 FF 25 4C 10 40 00 FF 25 DC 10  ..@..%..@..%L.@..%..
00011A8  40 00 FF 25 E0 10 40 00 FF 25 54 10 40 00 FF 25 E4 10 40 00  @..%..@..%T.@..%..@.
00011BC  FF 25 1C 10 40 00 FF 25 98 10 40 00 FF 25 08 10 40 00 FF 25  .%..@..%..@..%..@..%
00011D0  0C 10 40 00 FF 25 78 10 40 00 FF 25 C0 10 40 00 FF 25 94 10  ..@..%x.@..%..@..%..
00011E4  40 00 FF 25 A8 10 40 00 FF 25 9C 10 40 00 FF 25 B4 10 40 00  @..%..@..%..@..%..@.
00011F8  FF 25 04 10 40 00 FF 25 90 10 40 00 FF 25 8C 10 40 00 FF 25  .%..@..%..@..%..@..%
000120C  68 10 40 00 FF 25 84 10 40 00 FF 25 B0 10 40 00 FF 25 00 10  h.@..%..@..%..@..%..
         40 00 FF 25 38 10 40 00 FF 25 58 10 40 00 FF 25 30 10 40 00  @..%8.@..%X.@..%0.@.
         FF 25 50 10 40 00 FF 25 28 10 40 00 FF 25 44 10 40 00 FF 25  .%P.@..%(.@..%D.@..%
0001248  CC 10 40 00 68 20 14 40 00 E8 F0 FF FF FF 00 00 00 00 00 00  ..@.h .@............
000125C  30 00 00 00 68 00 00 00 40 00 00 00 BF BE B9 4E 1F 3B 0D 4C  0...h...@......N.;.L
0001270  87 DD A6 E2 F5 80 D1 B5 00 00 00 00 00 00 01 00 00 00 41 00  ..................A.
```

*Image 7. Insertion of the file that was not detected as malware between ranges.*

In this case it is possible to select one of the ranges that were obtained, considering range 4.061- 5.649. Creating then files of 1 byte, analyzing them as the image 8 shows. But in this case all of those that were not detected are subject to functionality tests.



¡AMENAZA DETECTADA!

Amenazas

Elige la acción a realizar para cada resultado y haz clic en "Aplicar".

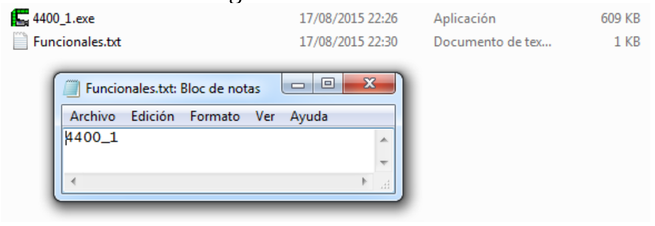
| Nombre de archivo | Gravedad | Estado | Acción |
|---|---|---|---|
| C:\Users\pruebas\Desktop\...\4635_1.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\4638_1.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\4641_1.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\4647_1.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\4653_1.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |
| C:\Users\pruebas\Desktop\...\4659_1.exe | Alto | Amenaza: Win32:Malware-gen | Eliminar |

*Image 8. Range of 1 byte of detected offset.*

It is worth noting that each one of the analysis performed by the antivirus was made under signature based detection and not using other detection techniques as the ones based on behavior or something similar.

## C. Functionality test

It consists on evaluating the functionality of the obtained files. The annotator is used for that and it should produce a file with extension TXT, with the same name than the executable one, which means that it keeps its operating properties just like the image 9 shows.



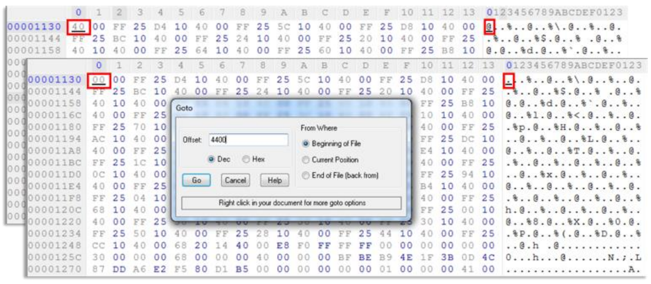
| | | | |
|---|---|---|---|
| 4400_1.exe | 17/08/2015 22:26 | Aplicación | 609 KB |
| Funcionales.txt | 17/08/2015 22:30 | Documento de tex... | 1 KB |

Funcionales.txt: Bloc de notas

4400_1

*Image 9. Annotator test functionality.*

On the image 9 it is possible to see that the offset 4.400 was not detected by the antivirus and keeps its functionality. This offset is the one that needs to be modified from the base malware, as the image 10 shows, to be analyzed later and prove its undetection, as the image 11 presents.



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10 11 12 13  0123456789ABCDEF0123
00001130  40 00 FF 25 D4 10 40 00 FF 25 5C 10 40 00 FF 25 D8 10 40 00  @..%..@..%\.@..%..@.
00001144  FF 25 BC 10 40 00 FF 25 24 10 40 00 FF 25 20 10 40 00 FF 25  .%..@..%$.@..% .@..%
00001158  40 10 40 00 FF 25 64 10 40 00 FF 25 60 10 40 00 FF 25 B8 10  @.@..%d.@..%`.@..%..
```

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10 11 12 13  0123456789ABCDEF0123
00001130  00 00 FF 25 D4 10 40 00 FF 25 5C 10 40 00 FF 25 D8 10 40 00  ...%..@..%\.@..%..@.
00001144  FF 25 BC 10 40 00 FF 25 24 10 40 00 FF 25 20 10 40 00 FF 25  .%..@..%$.@..% .@..%
00001158  40 10 40 00                                     FF 25 B8 10  @.@..%d.@..%`.@..%..
0000116C  40 00 FF 25                                     10 10 40 00  @..%l.@..%<.@..%..@.
00001180  FF 25 70 10                                     40 00 FF 25  .%p.@..%H.@..%..@..%
00001194  AC 10 40 00                                     FF 25 DC 10  ..@..%..@..%L.@..%..
000011A8  40 00 FF 25                                     E4 10 40 00  @..%..@..%T.@..%..@.
000011BC  FF 25 1C 10                                     40 00 FF 25  .%..@..%..@..%..@..%
000011D0  0C 10 40 00                                     FF 25 94 10  ..@..%x.@..%..@..%..
000011E4  40 00 FF 25                                     B4 10 40 00  @..%..@..%..@..%..@.
000011F8  FF 25 04 10                                     40 00 FF 25  .%..@..%..@..%..@..%
0000120C  68 10 40 00                                     FF 25 00 10  h.@..%..@..%..@..%..
00001220  40 00 FF 25                                     30 10 40 00  @..%8.@..%X.@..%0.@.
00001234  FF 25 50 10 40 00 FF 25 28 10 40 00 FF 25 44 10 40 00 FF 25  .%P.@..%(.@..%D.@..%
00001248  CC 10 40 00 68 20 14 40 00 E8 F0 FF FF FF 00 00 00 00 00 00  ..@.h .@............
0000125C  30 00 00 00 68 00 00 00 40 00 00 00 BF BE B9 4E 1F 3B 0D 4C  0...h...@.......N.;.L
00001270  87 DD A6 E2 F5 80 D1 B5 00 00 00 00 00 00 01 00 00 00 41 00  ...................A.
```

Goto

Offset: 4400

[✓ Dec] Hex

From Where

[✓ Beginning of File]

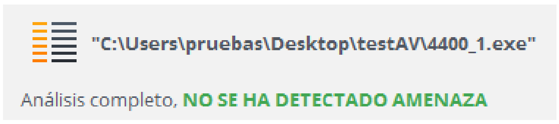
Current Position

End of File (back from)

Go Cancel Help

Right click in your document for more goto options

Image 10. *Modification offset original malware.*



"C:\Users\pruebas\Desktop\testAV\4400_1.exe"

Análisis completo, **NO SE HA DETECTADO AMENAZA**

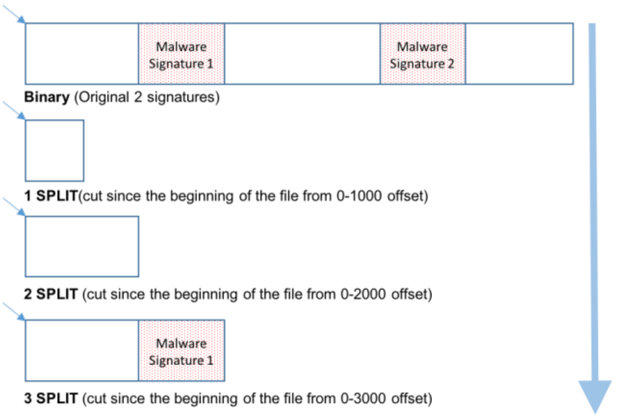
*Image 11. Evasion tests of the base malware.*

## IV. DEAD CODE INSERTION PROCEDURE TO MULTIPLE SIGNATURES: DSPLIT, AVFUCKER

When the AVFUCKER dead code insertion does not produce any result (functional and undetected files) by the end of its processing, it is necessary to evaluate if the file that needs to be undetected has multiple detectable signatures inside its structure.

Even though the AVFUCKER dead code insertion is functional for cases where there is only one signature on the file that needs to be cleaned, it is not when such file has on its hexadecimal structure more than one signature identified by the antivirus systems. In these cases it is required to perform another procedure so the dead code insertion can be made this time, detecting the exact areas (signatures) that are classified as malware, and work on these in an isolated way.

In order to keep malware that has more than a signature detected by the antivirus without being visible it is necessary to combine the DSSPLIT (find signatures) and AVFUCKER (insert dead code) techniques. To do so malware needs to be divided into pieces gradually so the signature can be found as the following image shows.



Malware Signature 1
Malware Signature 2
**Binary** (Original 2 signatures)
**1 SPLIT**(cut since the beginning of the file from 0-1000 offset)
**2 SPLIT** (cut since the beginning of the file from 0-2000 offset)
Malware Signature 1
**3 SPLIT** (cut since the beginning of the file from 0-3000 offset)

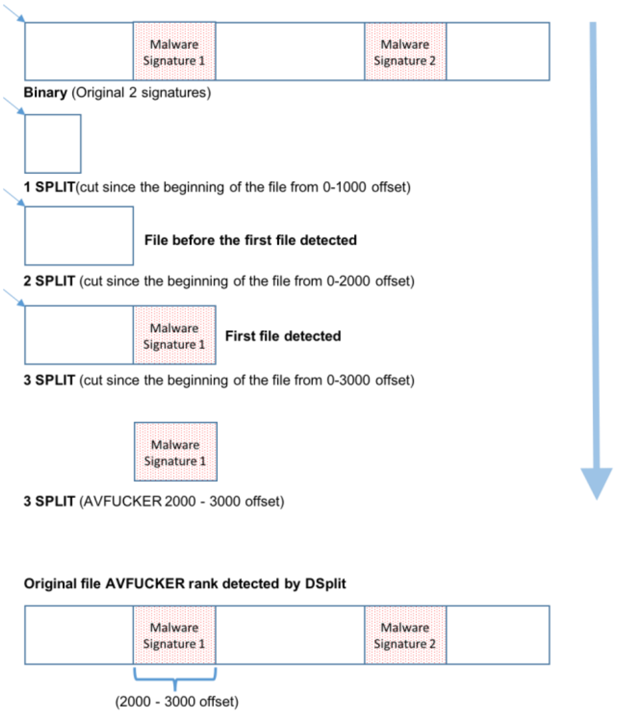
*Figure 8. DSPLIT procedure with multiple signatures.*

The original binary code is divided from the offset 0 to offset 1000, creating a new file; from offset 0 to offset 2000 for another file; from the offset 0 to offset 3000 creates another file. This happens successively producing both files and offsets that determine the original file. This way each new file would have a bigger size, since it has more information on its binary structure. Besides, these files will not be functional because they do not have the complete structure from the original source file.

Finding signatures

When analyzing the new samples, it is necessary to locate the first detected file (with the lowest offset range) and the previous undetected one, since they are the limits of the piece of code identified as the antivirus first signature, which then is going to go through the AVFUCKER procedure to clean this signature.

As the image presents, the first detected file corresponds to the piece from the offset 0 to 3000; and the previous undetected file goes from offset 0 to 2000, so the first signature is found between range 2000 and 3000. From now on, only this offset piece would be considered when obfuscating (insert dead code), since this file area is detected by the antivirus as the first signature.



Malware Signature 1
Malware Signature 2
**Binary** (Original 2 signatures)
**1 SPLIT**(cut since the beginning of the file from 0-1000 offset)
**File before the first file detected**
**2 SPLIT** (cut since the beginning of the file from 0-2000 offset)
Malware Signature 1
**First file detected**
**3 SPLIT** (cut since the beginning of the file from 0-3000 offset)
Malware Signature 1
**3 SPLIT** (AVFUCKER 2000 - 3000 offset)
**Original file AVFUCKER rank detected by DSplit**
Malware Signature 1
Malware Signature 2
(2000 - 3000 offset)

*Figure 9. DSPLIT signature location.*

If this process is performed on the original file, changing the values (AVFucker) on the area identified as the first signature, the final file would always be considered as malware since there are more pieces of codes on other file areas that are identified as malware. Thus, the final cleaning procedure is effective if both procedures are combined (AVFucker y DSplit).

Two files must be used to erase the detected signature:

a) Non-functional file: (piece identified as malware) this file is not functional because it covers only the hexadecimal information from the original file corresponding to offset 2000 to offset 3000.

b) Functional original file: (more than one signature identified as malware within its binary structure).

First, the AVFucker procedure must be applied on the file “a” (until getting 1 byte) and the samples must be analyzed; only the files that the antivirus does not classify as malware would remain. These samples are not functional since the file that they come from is just a piece of the original binary code file. As an example, the result of the antivirus analysis, as the image shows, has categorized 4 undetected files as malware that should be noted since they show the values (modified offset) that break the first signature. These values will be compared with the obtained results when applying AVFucker to the file B.

Now the AVfucker will be applied on the file “b” but only on the detected range on the first signature (from the offset 2000 to 3000). If results are analyzed by the antivirus all of them would be classified as malicious, since the second signature will be always detected within the binary structure. This is when the previous results must be compared so the necessary changes are then made. As the following image shows, it is possible to find 4 undetected files as malware (results “a”), only these 4 offsets will be used to modify the file “b” (creating a copy of the original for each offset that needs to be modified). Doing the functioning tests it is possible to indicate that only the offset (binary 5) keeps the integrity of the original binary file, so a new file is generated considered as file “c”, which has not the first signature but still has a second detectable signature by the antivirus.

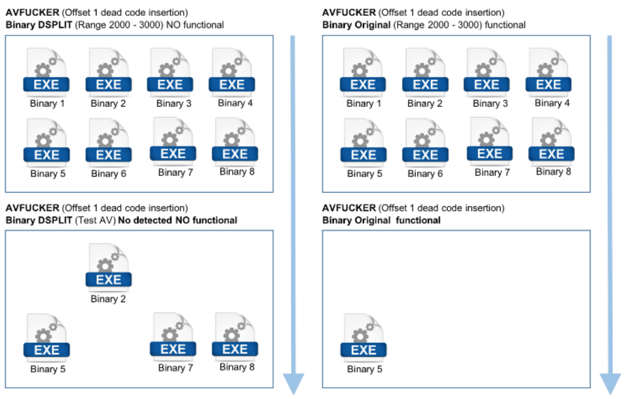

AVFUCKER (Offset 1 dead code insertion)
Binary DSPLIT (Range 2000 - 3000) NO functional
EXE Binary 1
EXE Binary 2
EXE Binary 3
EXE Binary 4
EXE Binary 5
EXE Binary 6
EXE Binary 7
EXE Binary 8
AVFUCKER (Offset 1 dead code insertion)
Binary Original (Range 2000 - 3000) functional
EXE Binary 1
EXE Binary 2
EXE Binary 3
EXE Binary 4
EXE Binary 5
EXE Binary 6
EXE Binary 7
EXE Binary 8
AVFUCKER (Offset 1 dead code insertion)
Binary DSPLIT (Test AV) No detected NO functional
EXE Binary 2
EXE Binary 5
EXE Binary 7
EXE Binary 8
AVFUCKER (Offset 1 dead code insertion)
Binary Original functional
EXE Binary 5

*Figure 10. DESPLIT procedure results,*

To find the second signature, the same previous logic must be used but this time from the file “c”, as the image shows. The second signature has been found between offset 4000 to offset 5000.

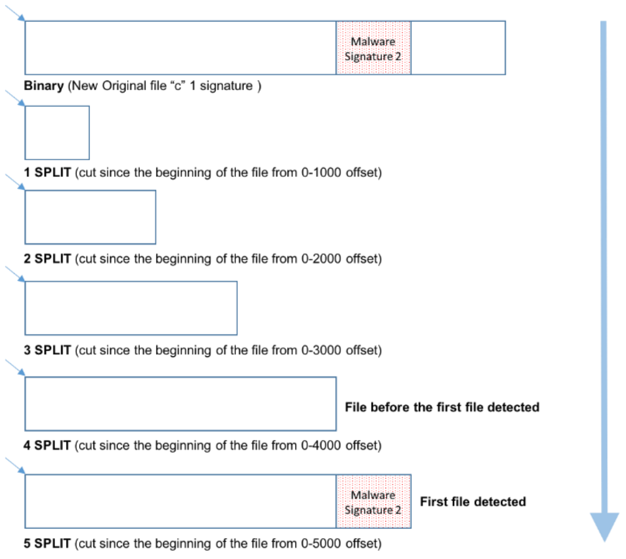

Malware Signature 2
Binary (New Original file “c” 1 signature )
1 SPLIT (cut since the beginning of the file from 0-1000 offset)
2 SPLIT (cut since the beginning of the file from 0-2000 offset)
3 SPLIT (cut since the beginning of the file from 0-3000 offset)
File before the first file detected
4 SPLIT (cut since the beginning of the file from 0-4000 offset)
Malware Signature 2
First file detected
5 SPLIT (cut since the beginning of the file from 0-5000 offset)

*Figure 11. DSPLIT second signature location.*

.

Just like the previous procedure, the DSplit technique application on the file allows to indicate the detected range as the second signature and then use AVFucker to this new piece of binary code.

## V. CONCLUSIONS

The obfuscation technique application to malware involves several aspects to consider when avoiding an antivirus that turn out to be a challenge for the modders world, since they must deal with the ability of the technologic equipment that they have, the time needed to perform the tests with the different antivirus engines and the maintenance of the malware functionality, crucial factor, to carry out the task.

This is how the present research has proposed a flow chart that proves through the use of tools, incorporated in the obfuscation procedure performance that it is possible to optimize the technical, economic, functional and timing resources on the malware manual update.

This means a real contribution not only for modders but for the Information Security Management Systems from different organizations, since it allows making evaluations to implemented security policies. For future action it is considered to develop a system that automates manual update that the obfuscation procedure execution requires.

## REFERENCES

[1] B. Mishra y A. Prajapatib, «Modelling and Simulation: Cyber War,» Elsevier Ltda, India, 2013.

[2] J. Carr, Inside Ciber Warfare, O'Reilly Media Inc, 2010.

[3] J. Z. Kolter y M. A. Maloof, «Learning to Detect and Classify Malicious Executables in the Wild,» Journal of Machine Learning Research 7, USA, 2006.

[4] C. Ammann, «Implementation of a PE-Crypter,» Nullsecurity, 2012.

[5] V. Tasiopoulos y S. Katsikas, «Bypass antivirus detection with encryption,» ACM, New York - USA, 2014..

[6] Blau, «Indetectables,» 12 Septiembre 2014. [En línea]. Available: http://www.indetectables.net/viewtopic.php?f=8&t=51265. [Último acceso: 30 Noviembre 2015].

[7] M. Egele, T. Scholte, E. Kirda y C. Kruegel, «A Survey on Automated Dynamic Malware Analysis Techniques and Tools,» ACM Computing Surveys, Vienna, 2014.

[8] D. Mohanty, «Anti-Virus Evasion Techniques and Countermeasures,» 2004.

[9] Z0MBiE, «Automated Reverse Engineering: Mistfall Engine,» Moscow, 2001.

[10] Teso, «Packet Storm,» 24 Diciembre 2002. [En línea]. Available: https://packetstormsecurity.com/files/30648/burneye-1.0.1-src.tar.bz2.html. [Último acceso: 29 Noviembre 2015].

[11] B. Bashari Rad, M. Masrom y S. Ibrahim, «Camouflage in Malware: from Encryption to Metamorphism,» IJCSNS International Journal of Computer Science and Network Security, VOL.12 No.8, 2012.

[12] A. Singh, «Anti-virus Evasion Techniques,» 2011.

[13] D. Cordero, C. D. Barría, C. Cubillos y C. Collazos, «Obfuscation method for the manual update of Malware, based on Crypter: A Survey,» En proceso de publicación, Santiago, 2015.

[14] A. Balakrishnan y C. Schulze, «Code Obfuscation Literature Survey,» University of Wisconsin, Madison, 2005.

[15] Z. Bazrafshan , H. Hashemi, S. Fard y A. Hamzeh, «A survey on heuristic malware detection techniques,» in Information and Knowledge Technology (IKT), 2013 5th Conference on , vol., no., pp.113-120, 28-30, Shiraz, 2013.

[16] K. Murad, S.-u.-H. Shirazi, Y. Zikria y N. Ikram, Evading Virus Detection Using Code Obfuscation, Berlín: Springer, 2010.

[17] A. Pasamar, «securitybydefault,» 02 Septiembre 2013. [En línea]. Available: http://www.securitybydefault.com/2013/09/crypters-localizando-firmas-de-los.html. [Último acceso: 29 Noviembre 2015].

[18] Pink, «Indetectables,» 04 Abril 2013. [En línea]. Available: http://www.indetectables.net/viewtopic.php?f=12&t=45686. [Último acceso: 29 Noviembre 2015].

[19] A. S. Inc, «Avast antivirus,» 15 Noviembre 2015. [En línea]. Available: https://www.avast.com/es-ww/index. [Último acceso: 29 Noviembre 2015].

[20] Mingo, «Indetectables,» 26 Enero 2011. [En línea]. Available: http://www.indetectables.net/viewtopic.php?f=&t=29725. [Último acceso: 29 Noviembre 2015].

[21] B. Software, «Hex Workshop,» 15 Noviembre 2015. [En línea]. Available: http://www.hexworkshop.com. [Último acceso: 29 Noviembre 2015].