J Comput Virol Hack Tech (2013) 9:171–178
DOI 10.1007/s11416-013-0192-5

INVITED PAPER

A study on common malware families evolution in 2012

Marius Barat · Dumitru-Bogdan Prelipcean · Dragoş Teodor Gavriluţ

Received: 1 July 2013 / Accepted: 11 September 2013 / Published online: 2 October 2013

Abstract With the exponential growth of malware in the last 5 years, the number of polymorphic malware increased as well. The aim of this paper is to describe the evolution throughout a year of four major malware families (FakeAlert, Sirefef, ZBot and Vundo). The analysis has been made in terms of polymorphic mechanisms with regards to the polymorphic mechanisms (such as changes in the packer module, changes in the geometry of file, variation of version information from the resource directory or different methods used to modify the icon of one file) which have been used in order to avoid their detection by anti-malware systems. The malware files were collected every week throughout one year's time. For each family we have recorded the new variants and the updates that were added to the old ones in order to avoid detection. We have managed to examine more than 1,000 new versions of such files. The current article includes an additional study case. The latter focuses on the methods that have been used by the FakeAlert malware family in order to modify their icons.

1 Introduction

In the last half of decade the number of malware has increased significantly. The latest reports from AvTest [1] show that more than 30 millions of new malware appeared in 2012 (with an increase of more than 40 % comparing to 2011). Most of the new malware are automatically generated. This also means that the number of polymorphic malware has increased as well. This survey follows the evolution of four major malware families (FakeAlert, Sirefef, ZBot and Vundo) over a period of one year. Since it is rather difficult to predict the exact moment a new malware is release into the wild, these families have been tracked on weekly basis. Different clustering techniques have been used to cluster the malware into families. For each malware family, this paper analyzes the methods they use to avoid detection. The most common ones are changing the packer, modifying the geometry of the file, adding or modifying file resources and so on. Over the one year period more than 1,000 new versions of these four families appeared (this means an average of more than 3 new versions every day). This paper is structured as follows: Chapter 3 describes the methods used to collect malware and a short description for each malware family. Chapter 4 describes the clustering method we have used to identify new versions of malware and filter files from different families. Finally, chapter 5 presents the observation we have made on how malware from this families modifies themselves in order to avoid detection. At the end of this chapter, some techniques for icon modification used in FakeAlert family are presented.

M. Barat (✉) · D.-B. Prelipcean · D. T. Gavriluţ
"Alexandru Ioan Cuza" University of Iasi, Romania, Iasi, Romania
e-mail: mbarat@bitdefender.com

M. Barat · D.-B. Prelipcean · D. T. Gavriluţ
Bitdefender Anti-Malware Laboratory, 37 Sfantul Lazar Street, Solomon's Building, Iasi, Romania
e-mail: bprelipcean@bitdefender.com

D. T. Gavriluţ
e-mail: dgavrilut@bitdefender.com

2 Related work

The malware history from the first viruses to the current malicious programs is an area of interest for the information security researchers. The purpose of these studies is to extract important information in order to improve the detection and protection methods.

The study of malicious software evolution is difficult due to the large number of instances and the complexity of this kind of computer programs so that in some cases can be done only for macroscopic observation. An empirical study on malware evolution was done by Gupta et al. [4] on metadata from malware instances with the purpose of classifying those instances in families and finding relations among the established families. The whole process is based on mining information from technical description of malware instances. Another study that that has been made focuses on the way in which malicious programs can evolve in order to bypass the detection methods used by anti-virus software [6].

For some specific families of malware there are statistics which show their evolution over a given length of time [2]. These statistics are usually computed based on the number of spread files, on location of those files or based on the impact over the users. There are also studies on for specific malware in which is presented a history, main purposes and properties [10]. Other researchers have studied the Internet content such as popular sites and social media offering an overview on the propagation and distribution of malware [13]. Reports from information security companies resumed the threats over a period and gave predictions about new trends in malicious activity [8,9,11].

Our work is similar to the previous enumerated and on top of that, we give results based on information extracted from malware samples and their methods of avoiding detection.

A different approach on the idea of malware evolution is given by Iliopoulos et al. [5] in which the darwinian model of evolution is adapted for studying malicious software. The latest studies focus on the mobile malware [12] and the social media with their impact.

3 Collecting samples

Unlike many years ago when one of the main purpose of a malicious programs developer was to create malicious software as proof of concepts, nowadays they are more and more tempted by the huge financial gains they could reach using such programs. This fact led to an exponential growth of the number of malicious files, determining anti-virus producers to research for new more proactive detection methods. One of the most difficult challenges in building such detections consists in finding strong enough detection models in order to detect as many new versions of a malicious program as possible.

Regarding the most prevalent malware families, there is a continuous process that includes both malicious software creators which modify their programs in order to bypass antivirus detections and also the anti-virus vendor trying to develop more proactive detections for those families.

Through this paper we want to provide a detailed statistic for the frequency with which the most prevalent malware families provide updates for an existing version, as well as the frequency with which they build new malware versions. Our study also presents the most common changes that a malware creator performs in order to bypass an existing anti-virus detection.

We have included in our study files that have appeared during the year 2012. In order to choose the most prevalent malware families we have monitored a set of 10 well-known malware families in the first 2 months of 2012. Based on the amount of new files gathered, we have chosen to analyse 4 malware families during the whole year, which are briefly mentioned further [3]:

Sirefef is a family of malware that uses rootkit techniques to hide its presence on an infected computer. As the main feature of this threat is hiding the payload may vary much from a version to another. Some of the payload purposes are: downloading and launching another components which can interfere with the user's Internet experience by filtering, redirecting and modifying internet traffic. Other components are clickers or they perform Bitcoin mining. It disables security features from the operating system such as the Windows Security Center Service, Windows Defender Service, Windows Firewall Service. It contacts and sends obtained information to various hosts. These hosts are usually servers controlled by the malware creators or users (people or organizations that gain benefits by controlling the malware).

Vundo is a malware family mainly used for the delivery of pop-ups and adverts to different websites which also have malicious content, usually rogue anti-viruses. Beside the main purpose, this threat can also download and execute arbitrary components with a large variety of payloads. The malware is usually installed as a Browser Helper Object.

FakeAlert is a large family of rogue anti-virus with a prolific activity and updates. This type of malware has the main goal to obtain financial revenues by convincing the user that his computer is infected with many viruses. Once this is achieve, different disinfection modules are sold to the user to help him/her get rid of non-existing malware.

Zbot (Zeus) is a malware which is used to steal information about bank accounts and credentials. This malware is also wide spread with the biggest network of infected computers. We considered in our study this threat due to the big amount of updates and new versions which make this trojan difficult to detected and clean.

The set of files used for this paper consists of files from the Bitdefender malware data set, gathered in 2012, most of them being downloaded from a continuously updated data set of urls known to spread malicious files. In order to ensure these files belong to one of the 4 monitored families, we manually analyzed these files. Due the fact that the timestamp from the file header is not relevant to be considered as the date when those file appeared and because there is the possibility that the malware appeared on a difference of time between the time we downloaded a sample and the time it became available through a specific url, we decided to split the whole year in weeks. We consider that a file appeared in a certain week, instead of specifying it appeared in a certain day of the year.

Considering that our goal is to find how often new versions of malware families appeared, or how often it updates an existing versions, we will use in our purpose only those files which when were first seen, were not detected.

4 Clustering samples: new malware versions versus old version updates

At this point, we have the set of files manually analysed and assigned to one of the 4 malware families and for each sample we know the week it has first appeared. The next step we perform is to decide for each sample if it is a modified version of an old sample, or a brand new one.

Our approach regarding this issue consists in clustering the files that belong to the same malware family, each of the resulting clusters describing a certain malware version. For those clusters that contain files gathered in different weeks, those of them which have appeared in the first week when files from the cluster have been seen, are considered to be the appearance of a new version of those malware, while the rest of files which have appeared in other weeks are considered to be updates of the same version.

For each sample we have defined a set of 6,125 Boolean attributes: header information, imported functions, exported functions, compiler, file packer, resource information, dynamic characteristics like starting a processess, creating files, modifying registries and many other characteristics depending on the file type.

Having extracted all these attributes, we computed the distances between files that belong to the same malware family using two metrics:

- The Manhattan Distance [7]

  n $\leftarrow$ total number of features
  F_i,F_j are two files, each one described by a set of attributes
  F_i.Features, respectively F_j.Features
  $F_i.\mathrm{Features}_k$ $\leftarrow$ value for the k-th attribute for the F_i file
  $F_j.\mathrm{Features}_k$ $\leftarrow$ value for the k-th attribute for the F_j file
  $$ManhattanDistance(F_i, F_j) = \sum_{k=1}^{n} Abs(F_i.Features_k - F_j.Features_k)$$

- Manhattan weighted distance
  Since not all the features have the same relevance in defining a file's behavior and as some of them are very likely to be changed at each malware update, it is useful not to use all the features with the same weight

  n $\leftarrow$ total number of features
  w $\leftarrow$ $\{w_1, w_2, \dots w_n\}$ a set of choosen weights for each feature
  F_i, F_j, $F_i.\mathrm{Features}_k$, $F_j.\mathrm{Features}_k$ are the same described above
  $$WeightedManhattanDistance(F_i, F_j) = \sum_{k=1}^{n} w_k \times Abs(F_i.Features_k - F_j.Features_k)$$

We used this metric to compute distances associating different weights to the features according to their relevance: features related to the file's geometry characteristics, which are very likely to be changed at consecutive updates, were given smaller weights, while the dynamic features, which better describe the behavior for a sample received bigger weights.

Using this approach, we consider that two different samples are similar (which would assign them to the same malware version) if the computed distance between them is smaller than an established threshold. We propose the following method to choose the threshold:

$$MinValue \leftarrow \text{the smallest distance between two instances}$$
$$MaxValue \leftarrow \text{the largest distance between two instances}$$
$$T \leftarrow MinValue + \frac{x}{100}(MaxValue - MinValue)$$

For our study, after plotting the histograms for the distances computed using the metrics presented above, we have decided to compute the threshold using the formula we have just presented with the parameter x set to 5.

Using the proposed clustering algorithm (Algorithm 1), we clustered the samples for each malware family and the number of resulted clusters for each of the two metrics used are presented in Table 1.

Algorithm 1 Distances based clustering algorithm

```
1: $C \leftarrow \emptyset$                                   ▷ $C$ stands for the clusters set
2: for all $(f1, f2)$ do                                    ▷ $f1, f2$ stands for two files
3:     if $distance(f1, f2) <= treshold$ then
4:         if $((\forall i \in \overline{0 \dots |C|}: f1 \notin C_i)$ and $\forall i \in \overline{0 \dots |C|}: f2 \notin C_i))$ then
5:             C.Append(NewCluster($f1, f2$))
6:         else if $((\exists i \in \overline{0 \dots |C|} : f1 \in C_i)$ and $(\forall j \in \overline{0 \dots |C|} : f2 \notin C_j))$ then
7:             Assign($f2, C_i$)
8:         else if $((\exists i \in \overline{0 \dots |C|} : f2 \in C_j)$ and $(\forall j \in \overline{0 \dots |C|} : f1 \notin C_j))$ then
9:             Assign($f1, C_i$)
10:        else if $((\exists i \in \overline{0 \dots |C|} : f2 \in C_i)$ and $(\exists j \in \overline{0 \dots |C|} : f2 \in C_j)$ and $i \neq j)$ then
11:            MergeClusters($C_i, C_j$)
12:        end if
13:    end if
14: end for
```

We applied the presented clustering algorithm and then manually analysed resulted clusters. The method that presents higher accuracy regarding the assignment of files to a certain cluster was the one that uses the ManhattanWeighted

Table 1 Number of resulted clusters

Malware family	Manhattan	ManhattanWeighted
FakeAlert	287	312
Zbot	495	474
Sirefef	198	160
Vundo	110	133

metric. The clustering performed using the Manhattan metric is penalized by the fact that the largest clusters created actually includes more than one version of a malware family. This is the reason why we have chosen to use further in our study the clusters obtained using the ManhattanWeighted metric.

Using these clusters and also the week of the year when each sample appeared, we plotted two graphics in order to highlight the frequency with which new malware versions appear and also how often they are updated.

Figure 1 presents the number of new versions that appeared for a malware family in every week of the 2012 year. We started collecting samples on 01.01.2012 and this led us to consider that each generated cluster that contains files that appeared in the first week of 2012 represents a new version of those malware family. This is the reason why in the first week all the malware families are present through a larger number of new versions than the other weeks of the year as shown in the Fig. 1. This is why in the first week of the year the malware families had no updates to an existing version as being shown in Fig. 2.

Figure 2 present the number of updates that malware creators provide for existing versions of a malware family on every week of the year.

From Figs. 1 and 2 can be observered that the Zbot family is the one that is most active in terms of providing updates to existing versions, or even new malware versions, while the Vundo family spreads a considerably smaller amount of new binaries.

The Sirefef family, even if it spreads new malware versions at longer periods of time, it constantly updates the existing ones.

Regarding the Vundo family, it can be observed that in the first third of the year it was quite active, but in the rest of the year its presence was more discrete.

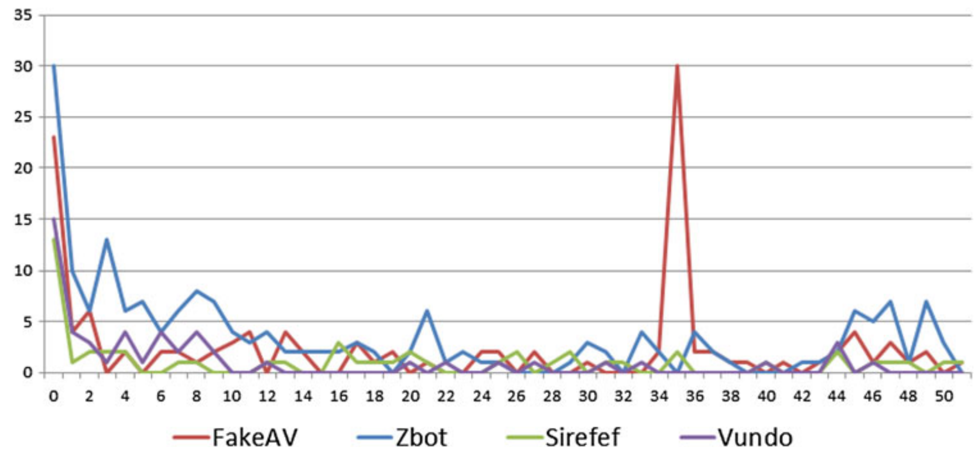


Fig. 1 New malware versions in 2012

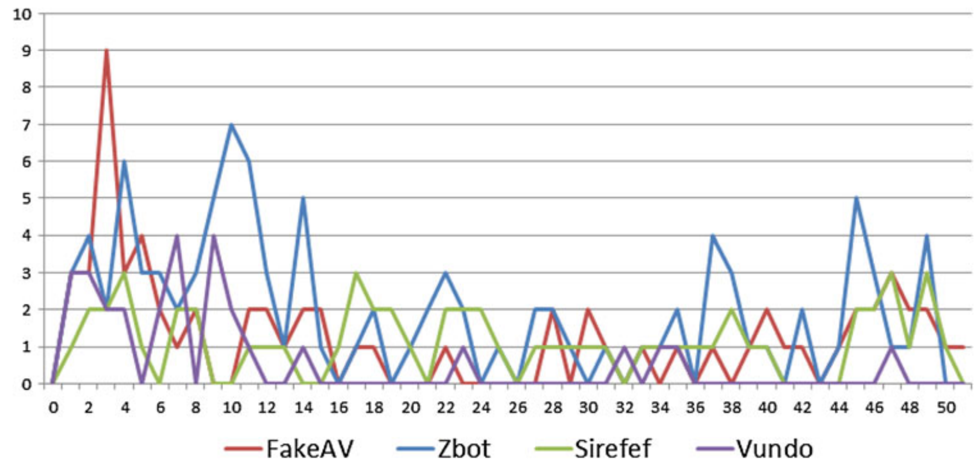


Fig. 2 Malware versions updates in 2012

Table 2 Sirefef asm code

```
.text:00419454              push ebp
.text:00419455              mov ebp, esp
.text:00419457              and esp, 0FFFFFFF8h
.text:0041945A              sub esp, 9Ch
.text:00419460              push ebx
.text:00419461              jmp loc_419532
.text:00419532  loc_419532:
.text:00419532              push esi                      ; lpLastAccessTime
.text:00419533              push edi                      ; lpCreationTime
.text:00419534              push 0                        ; dwFlags
.text:00419536              push offset CurrentChar       ; "rieuritrjkgfldglfdjghofdijhoidfjhgdfhgf"
.text:0041953B              jmp loc_41960C
.text:0041960C  loc_41960C:
.text:0041960C              push 0                        ; CodePage
.text:0041960E              call ds:CharNextExA
.text:00419614              mov al, [eax+1]
.text:00419617              cmp al, byte ptr CurrentChar+2
.text:0041961D              jz loc_41962B
```

5 Common detection evasion techniques

Each cluster generated in the previous step describes a certain version of a malware. We wanted to find out which are the most common modifications that the malware creators perform in order to bypass the anti-virus detection. This is very useful for the malware analysts when developing malware detections. Knowing this common modifications for each family can help the malware analyst to provide a more proactive detection, independent of the file characteristics which are most likely to be changed.

One of the most common techniques used by the malware creators in order to bypass the anti-virus detection consists of code obfuscation. Obfuscation makes the malicious file code more difficult to be analysed by a malware analyst, leading to an increased amount of time spent to analyse a sample. It is very difficult to distinguish automatically between the obfuscated code and the one that is not obfuscated. This causes many difficulties to those detections based on the instructions flow.

This technique is used by all of these malware families and it consists of using a custom made obfuscated file packer in order to protect the malicious code. Due to the diversity of randomly chosen instructions used to obfuscate the packer's code, many times the anti-malware emulators fails to unpack the malware's code.

Table 2 present the first instruction from the entry point of a sirefef sample. As it can be seen in the instructions flow, it uses two jumps, one of them just between the push instructions that transmit the parameters for the call instruction. At addres 0041960E, it calls the CharNextExA API, which being rarely used is not handled by most of the existing emulators. Due to this fact, for this sample, the emulation process ends at this call instruction, failing to unpack it. In a real enviroment, this code sequence will be executed normally and the sample will infect the host system.

Another common detection evasion technique consists of modifying the file geometry. This includes a wide range of changes that can be performed, including modifications regarding the number and size of sections, the number of directories, the number and the types of resources, the entry-point section, directory's sections, sections code entropy and so on.

Other modifications that are often met mainly at malware files that belong to the Zbot family or Vundo family are the random strings used for the section names, for the resource names, for the export names and also for the fields that are part of the file's version name. Many times these strings are generated using several patterns, which are frequently updated.

In Table 3 are presented 4 version info resources that belong to 4 samples from a Zbot version. Being built using the same pattern, they are generated in order to simulate a version info resource of a legitimate file. The "Company Name" field consists of the name of a known company. This field and the "Original File Name" field are the only ones that can contain words longer then 6 characters. The "Original File Name" has also the restrictions of not containing spaces and ends with the .exe extension. The "Internal Name", "Product Name" and "Legal Copyright" contains words no longer then 6 characters, all of them beginning with a capital letter. Moreover, the "Legal Copyright" field ends with two

Table 3 Version info resources

CompanyName: Packard Bell BV	CompanyName: Belkin Corporation
FileDescription: With Enigma Gnaws Brian Gash Spite	FileDescription: Piers Bogus View
FileVersion: 3.10	FileVersion: 8.8
InternalName: Brain Wyatt Beggar	InternalName: Quack Media Waldo
LegalCopyright: Wells Soggy Sends Avid 1995–2011	LegalCopyright: Royal (Whoa Fence) 1999–2008
OriginalFileName: Abase.exe	OriginalFileName: Tx5baojo5srl.exe
ProductName: Spoke Fizz Juice	ProductName: Zowie
ProductVersion: 3.10	ProductVersion: 8.8
CompanyName:Prolific Technology Inc.	CompanyName: Packard Bell BV
FileDescription: Lenny	FileDescription: Need Bush Realm Rayon
FileVersion: 1.9	FileVersion: 2.6
InternalName: Cried Spits Fop Emma	InternalName: Swing Puff Dally Meteor Lets Late
LegalCopyright:Retch Psi Wages Antic 2003–2005	LegalCopyright: Quake Pleas Perch Igor 2001–2006
OriginalFileName: Fad.exe	OriginalFileName: Lazy.exe
ProductName: Store Bye Jury Tent	ProductName: Womb Woes Work
ProductVersion: 1.9	ProductVersion: 2.6

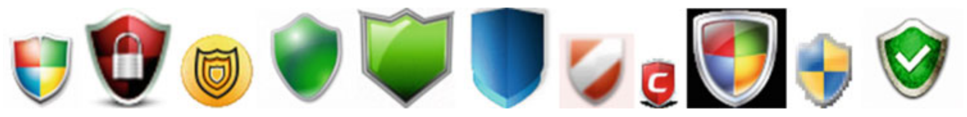


Fig. 3 Shield based FakeAlert icons

numbers representing two years, separated by a short line character.

A frequently used technique by the Zbot and FakeAlert creators consists of adding different unused resources to the binary, using different languages for them such as Nordic languages, Spanish languages, German language or French language.

A specific characteristic to the Sirefef family is that they steal the version info resource from legitimate files that belong to the Microsoft operating system and they also try to match as much as possible with the windows files, regarding the file geometry. When updating an existing version, the Sirefef's creators use to change the existing version info with another one stolen from another legitimate file that belongs to the operating system and tries to adopt its characteristics regarding the file geometry.

A characteristic found at versions that belong to each of the 4 families is the use of version info resources stolen from binaries that belong to legitimate applications. The malware creators usually use a stolen version info resource for a couple of days and then they update the malware changing its version info resource with another one stolen from a legitimate application. The resources theft is also implemented regarding the icon resources.

Another interesting update delivered by the malware creators for both Zbot and Sirefef malware during the 2012 year

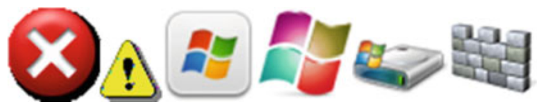


Fig. 4 Error/Windows like FakeAlert icons

consisted of adding digital signatures to their samples, even if most of the times it was invalid.

A polymorphic modification specific to the FakeAlert family is the very high frequency in which they change the samples icon. A more detailed study on this topic is presented in the next case study.

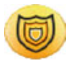
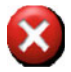
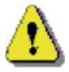
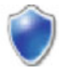
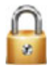
6 FakeAlert icons: case study

Most of the icons a malware of this type uses look like a shield with different variations. The following picture (Fig. 3) shows the list of some of the most common icon used in the last year by FakeAlert family.

Besides this type of icons other are used as well (usually related to error message / alert messages or different Windows like icons), shown in Fig. 4.

In each of these cases the icons were used for a large period of time with some modification to the image. Table 4 shows

Table 4 Most common icons in FakeAlert family and weeks when they changed

Icon	Weeks
	3, 4, 5, 7, 12, 35, 42, 46
	0, 1, 2, 3, 14, 48
	13, 14, 17, 24, 31, 37, 39, 40
	28, 30, 33, 34, 49
	37, 40, 41, 44, 45, 46, 47, 49

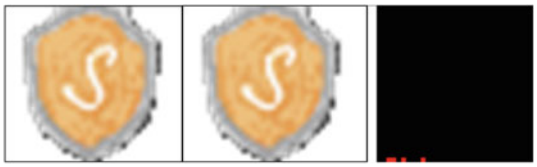


Fig. 5 Alpha channel difference

the most used icons and the weeks a new version of that icon have appeared.

Since most of the vendors have different methods that they can use for malware detection based on the icon, several methods are used in the FakeAlert family to adjust an icon in such a way so that it will look very similar to the original one (virtually identical to the naked eye) but different from the binary point of view.

- Adding some pixels with a very low Alpha channel (0 or 1). Since Alpha channel controls transparency (0 means completely transparent) then adding a pixel with the alpha channel closer to 0 will not render that colour at all. The two images shown below (Fig. 5) are magnified (4 times) and they look identically. However the last image of the set represents the difference between those images (a black point means that at that point both images are identical and a red point means that they are different). As it can be seen, the two pictures have exactly 4 pixels different (created with Alpha channel adjustments).
- Sharpening an image. This technique creates two different images (Fig. 6). While these images look different, when minimized they look very similar.
- Adjusting the RGB value of all or some of the pixel s from an icon. The idea is to select some pixel and adjust some of the RGB channels with a slightly different value (for example by adding small value like 1 or 2).

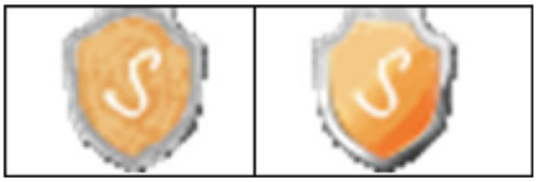


Fig. 6 Sharpening an image

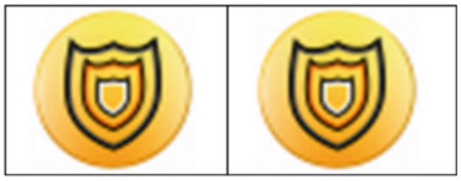


Fig. 7 Icons with slightly different value for some pixels

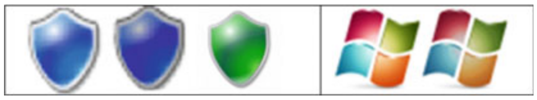


Fig. 8 Different shade icons

Even if the two icons in the previous image (Fig. 7) look identical, from the binary point of view they are completely different (only 7 pixels are identical on this two images - this means that more than 99.7 % of this images are different).

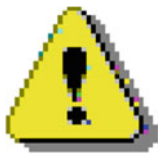
- Changing the shade of the same image (ex. Fig. 8). In some cases this is obvious to the naked eye as well, in other it is more difficult to observe.
- Add some pixels with a random colour to the image:

The previous image (Fig. 9) has been magnified 5 times. As it can be seen, some pixels with different colour were added to the image. However, in this case only 26 pixels were added (this means that less the 2.6 % of the image is modified). When dealing with very small images (this is in fact a 32 × 32 pixel icon, these modification are not visible to the naked eye).

7 Conclusions

In order to build a stronger proactive detection for a widespread malware family, a survey on its evolution is very use-

Fig. 9 Adding some extra pixels to an icon



ful. Finding the most common changes that malware creators perform in their binaries help the malware analyst in the development process of new detection mechanisms. This survey presents the results of an one year process of monitoring 4 of the most common malware families. Several statistics regarding the frequency the malware creators provide new binaries are presented for each of the 4 monitored families. Besides these statistics, the most common changes performed by the malware creators are highlighted, some of them being presented in more details.

One of the issues that had to be performed to make the data set of files usable for our purpose was to identify all different versions for a malware family. We have provided an automated mechanism to handle this issue using a distances based clustering algorithm. This clustering technique for malicious files can be successfully used in other purposes too, such as grouping files that have to be analysed by a malware analyst according to the similarities between them. In this way, he can analyse a group of files once, instead analysing a single file.

References

1. http://www.av-test.org/en/statistics/malware/
2. https://zeustracker.abuse.ch/statistic.php
3. http://www.microsoft.com/security/portal/
4. Gupta, A., Kuppili, P., Akella, A., Barford, P.: An empirical study of malware evolution. In Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First, International, pp. 1–10, jan. (2009)
5. Iliopoulos, D., Adami, C., Szor, P.: Darwin inside the machines: Malware evolution and the consequencs for computer security. CoRR, abs/1111.2503, 2011
6. Tonimir, K., Klasic, D., Hutinski, Z.: A multiple layered approach to malware identification and classification problem. In Procceding of the 21st Central European Conference on Information and Intelligent Systems, jul, 2010, pp. 429–433, (2010)
7. Krause, E.: Taxicab Geometry: an adventure in non-Euclidean geometry. Dover Publications, New York, pp. 1–5 (1987)
8. McAfee Labs. 2012 threat predictions. Technical report. http://www.mcafee.com/us/resources/reports/rp-threat-predictions-2012.pdf
9. Microsoft: The evolution of malware and the threat landscape—a 10-year review. Technical report (2012)
10. Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford-Chen, S., Weaver, N.: Inside the slammer worm. IEEE Secur Priv **1**(4), 33–39 (2003)
11. Juniper Networks: The evolving threat landscape. Technical report. http://www.juniper.net/us/en/local/pdf/whitepapers/2000371-en.pdf (2012)
12. Srikanth, R.: Mobile malware evolution, detection and defense. pp. 3–13 (2012)
13. Yan, G., Chen, G., Eidenbenz, S., Li, N.: Malware propagation in online social networks: nature, dynamics, and defense implications. In Cheung, B., S., N, Hui, L., C., K., Sandhu, R., S. and Wong, DS., eds, ASIACCS, ACM, pp. 196–206 (2011)