Classification and Detection of Metamorphic Malware using Value Set Analysis

Felix Leder, Bastian Steinbock, Peter Martini
University of Bonn
Institute of Computer Science IV
Roemerstr. 164, 53117 Bonn, Germany
Email: {leder, steinboc, martini}@cs.uni-bonn.de

Abstract

Metamorphic malware changes the structure of its code from infection to infection. This makes it very hard to classify or to detect. While the byte-sequence of two variants may be completely different, the core functionality of the malware has to stay the same. This includes the use of flags and constants that have to be consistent at specific points. We present a novel approach that allows us to detect metamorphic variants. Based on this detection, it is also possible to classify new samples to a metamorphic family. Our approach identifies variants by tracking the use of consistent values throughout the malware. Our evaluation shows a 100% detection rate with 0 false positives for all metamorphic samples that do not change their behavior.

1. Introduction

The number of new malware specimen has significantly increased over the last years and basically shows an exponential growth [13]. The vast amount of new specimen cannot be analyzed fast enough. Thus, many of the modern antivirus products are not able to keep up with the growing threat. Statistics from [9] show that less than 5% of all submitted malware samples were detected by all virus scanners.

The detection rate is even worse for metamorphic malware. Metamorphic malware changes its code's appearance and structure from infection to infection. It does so e.g. by inserting junk code, which does not change the behavior, by replacing instructions with equivalent operations, and by changing the order of code blocks. Metamorphism is also used in polymorphic malware. Here, the acutal malcode is encrypted and a decryptor is included in the executable. Polymorphic malware uses metamorphic decryptors to avoid the detection and classification by the decryptor [21]. One prominent example for fully metamorphic malware is Lexotan32 [8]. Even though, this virus has been around since 2002, only 12.6% out of our infected samples were identified by the 40 virus-scanners at [9]. None of those scanners was able to detect all of the samples.

Metamorphic malware can change its code and structure in a way that less than half a dozen bytes remain available for signatures, which is impossible to use for detection. While the code itself changes significantly, the general behavior of the malware and thus the actual malfunctionality stays the same. This is reflected by the data that is used inside the malware. Certain values like loop counters, parameters to external library functions, like the "protocol" parameter to *socket()*, or flags have to stay the same at specific logical points inside each variant. However, those values can be constructed by metamorphic malware in various ways. As an example, the instruction `mov eax, 5` leads to the situation in which a register holds the value 5. So do the following two code sequences.

```
mov ebx, 2
add ebx, 3

xor ecx, ecx
sub ecx, -5
```

In this paper, we present a novel approach to detect and classify metamorphic malware by extracting characteristic sets of values. This approach is based on a value set analysis [11], which is a static analysis technique that tracks the propagation and changes of values throughout an executable. This characteristic set is determined by a refinement step that creates an intersection of all sets of values found in multiple variants of the same malware. With this refinement step it is even possible to separate the values of file-infecting malware from those values originating from the host program's code. The actual identification is based on an algorithm that identifies the best matching position for each of the characteristic's value set inside a specimen. Different parameters and matching schemes are studied to achieve the best detection performance.

With our approach, we were able to perfectly identify and differentiate all of six metamorphic specimen without

any false negatives or false positives.

The rest of the paper is structured as follows. Section 2 gives an overview of both commercial and academic related work. Section 3 gives a brief introduction into value set analysis. Section 4 explains the general methodology of the identification process, the use of points of interest, and the matching. In section 5, the matching parameters are determined and the evaluation of our approach based on six malware specimen is presented. Section 6 concludes our achievements and summarizes the results.

2. Related Work

Methods and tools that detect and classify malware exist in a range of commercial products as well as in academic research. It has been proven that it is not possible to develop a detection system that can detect all types of malware without any false negatives and false positives [12].

This section presents an overview of both commercial and academic approaches and explains the deficiencies found especially in commercial products when trying to detect metamorphic malware.

2.1. Commercial Approaches

Commercial products incorporate a wide range of different scanning and classification techniques like checksumming, string scanning, smart scanning, X-Ray scanning, code emulation, geometric detection, and heuristics [21]. Since the complexity of these approaches increases and commercial products try to achieve real-time scanning, the use of filters is common. Examples of filters are the exclusion of files that don't have an ".exe" suffix or lack executable headers, like PE or ELF [19, 10]. Other filters are to scan only in the boot-sector for boot-sector-viruses. Means to speed up detection in AV include *Top-and-Tail scanning*, which only scans certain regions at the start and end of each file, as well as *Entry-Point scanning* that is performed around the entry point of a file. It is obvious that the types of filters have to be tailored specifically to malware families.

Check-summing is the fastest but least reliable technique. The file that is to be scanned is check-summed using cryptographic hash functions, like MD5 [20], and compared to a list of malicious programs as well as possible whitelists. A variant of this approach is to hash only parts of the file. A little more sophisticated is *string scanning*, which scans the considered files for common substrings that only found in specific malware. Both approaches are not able to detect strong metamorphic because of the high variability of the infections. The chances for detection even get worse for file-infecting malware that morphs into a host program. An extension to string scanning is the use of wildcards, similar to regular expressions, in the search string. This allows for little more flexibility and can detect slightly changing code but not sophisticated morphing. *Smart scanning* is a special form of wildcard scanning that leaves out irrelevant parts of the inspected file, like obvious junk code. It is able to detect malware that performs metamorphism through the insertion of junk code but fails for semantic code replacement or code reordering [7].

Since real malcode is often encrypted, commercial products often target the encryption of the malware's body and then apply one of the scanning methods described above. While those methods can be efficient for this scenario that do not improve the detection of metamorphic malware which does not hide its code because of its ever-changing nature. *X-Ray scanning* targets the encryption often used by attempting simple, standard decryption based on simple arithmetic operations. *Code emulation* executes the start of the malcode in a small virtual machine trying to find the end of the decryption routine included.

A third category of commercial products searches for typical anomalies in executable files. These can be changes to the host program of specific file-infecting malware or unusual layout inside the malicious programs themselves. If tailored to specific families, this detection is called *geometric detection*. It is often prone to false positives [21]. More generic detections are based on typical malcode *heuristics*, like entry points in the last section of the infected file, suspicious code flow redirections, or inconsistent file header value [19].

2.2. Academic Approaches

While most commercial approaches try to detect malware by scanning for signatures, several newer approaches aim to detect and classify malware by its behavior. MetaAware [22] is closest to our approach by examining the code and data flow between system functions. From those they derive patterns that are matched to a metamorphed variant. Based on this matching attempt a similarity score is computed. Our approach is not only based on system functions but on all parts of the possibly infected file. Furthermore, MetaAware does not use refinement to extract the malcode from files, in which it has been merged with host code.

The method presented in [14], uses *code normalization* to remove obfuscation techniques, which are typically used within metamorphism. After the normalization step, the files are classified using subgraph isomorphism matching of the inter-procedural control flow graph [17]. Different normalization techniques exist. [16] presents how to revert common obfuscation, namely junk insertion, code reordering, and packing. Related to [14] is the detection and removal of redundant and useless instructions [15], which is

based on compiler optimization and *clone detection* to identify duplicate code. While those approaches attempt to revert known obfuscation they are no generic means for metamorphic malware.

The approach presented in [18] creates disassemblies that are more robust against traditional obfuscation by performing a more sophisticated reachability analysis. The analysis checks the plausibility of the existence of concurrent code-blocks. While this improves readability and analysis, it cannot directly be used to detect metamorphism or to classify metamorphic malware.

3. Value Set Analysis

The value set analysis [11] is the foundation of our metamorphic detection approach. The value set analysis (VSA) is a static code analysis that estimates the memory contents at all instructions of a program without executing it. This is achieved by performing a detailed data flow analysis throughout the whole executable and tracking the propagation of possible values. The following example shows the general principle. Each line shows the x86 assembly instruction and the value set belonging to it.

```
loc_0x100:
   mov eax, 1          {eax=1}
   mov ebx, loc_0x100 {eax=1, ebx=0x100}
   add eax, ebx        {eax=0x101, ebx=0x100}
```

Values are tracked for registers, global memory, stack, and heap locations. They are named *memory locations* because of being data stores. The VSA tries to determine the contents for all memory locations at every instruction of the program. Unlike in dynamic analysis, in which concrete values are assigned to each memory location at a point in run-time, the static analysis has to approximate all the values that an instruction's operand may contain. Since some values, like host dependend information, can only be determined at run-time, an exact determination is sometimes not possible. In those cases the value set analysis performs an over-approximation such that the real values are definitely included in the determined range. This over-approximation is extensively used during the analysis of loops and recursions in order to speed up the analysis. To be memory efficient, the VSA performs two optimizations. First, memory locations for which either no values can be determined or that can contain all possible values are not stored, which is implicit storage. Furthermore, instead of storing the exact values, more memory efficient structures are used. One example in the use of *strided intervals* [11]. They are described by a lower bound (lb), upper bound (ub) and stride (s) between elements. The values are described by $s * [lb, ub]$. As an example, the values $\{4, 8, 10, 12, 14\}$ can be described by $2 * [4, 14]$.

The detection approach presented here is based on our own VSA implementation that differs slightly from the one presented in [11]. For efficiency reasons, only registers and stack values are used. If the number of values for a given memory location stays below a given threshold, the concrete values are stored. Strided intervals are only used when the number of values exceeds the threshold.

More formally, each instruction inside a program is mapped to a *value set*. Such a *value set* V is a collection of *data objects* (C_i):

$$V = \{C_1, C_2, ..., C_n\} \tag{1}$$

A single data object represents a memory location l and a set of possible values $\{s\}$ that memory location l may contain at the time when the instruction considered would be reached during execution:

$$C_i = (l_i, \{s_i\}) \tag{2}$$

For the value set analysis implemented, l can be either a specific processor register or a location on the virtual stack described by the its offset.

$$l \in \{\text{register},\ \text{stack location @ offset y}\} \tag{3}$$

The offset is the relative position compared to the position of the stack pointer at the beginning of the currently analyzed function. Stack locations mostly represent local variables or parameters for function calls. $\{s\}$ can be a set of concrete values or an over-approximation described by a strided interval.

4. Methodology

Two infections by the same metamorphic malware may have only few bytes of similar code in common. Thus, they are impossible to detect by regular string sequences without the risk of creating an unacceptable percentage of false positives. This becomes even more complicated for file-infecting viruses, in which the code of the infected host program is mixed with the malcode. Even though the code of the malware may change completely, the malfunctions contained inside the binary have to stay the same: The malware does only change its appearance but not its behavior. This includes that certain values or possible sets of values do not change from infection to infection. Simple examples are loops with a constant number of iterations, or calls to specific system functions like *socket()*. The constant have to be used at the same point in the logic of the malware, even though its code and structure looks completely different. The *socket()* function e.g. is typically used to create UDP (17) or TCP (6) sockets. Thus, the set of values on the stack just before the function call is $\{6, 17\}$. Our approach

exploits the fact, that certain sets of values stay constant among all metamorphed variants which allows us to classify and detect them. Figure 1 illustrates the idea for an example with slight metamorphic modifications. On the left is the original program, the right side is morphed. The marked instructions are junk and do not have any impact on the behavior. The points *O1* and *M1* mark identical points. The value set at those points is $\{eax = 10, ebx = 2, ecx = 5\}$. In the morphed variant, these values have been computed using intermediate calculations. Points *O2* and *M2* are inside a loop and therefore over-approximated. Still, the value sets contain the same over-approximations. $\{eax \geq 10, edx \leq 5\}$. While these code sequences look similar, their binary representation is completely different and cannot be used for reliable string signatures.

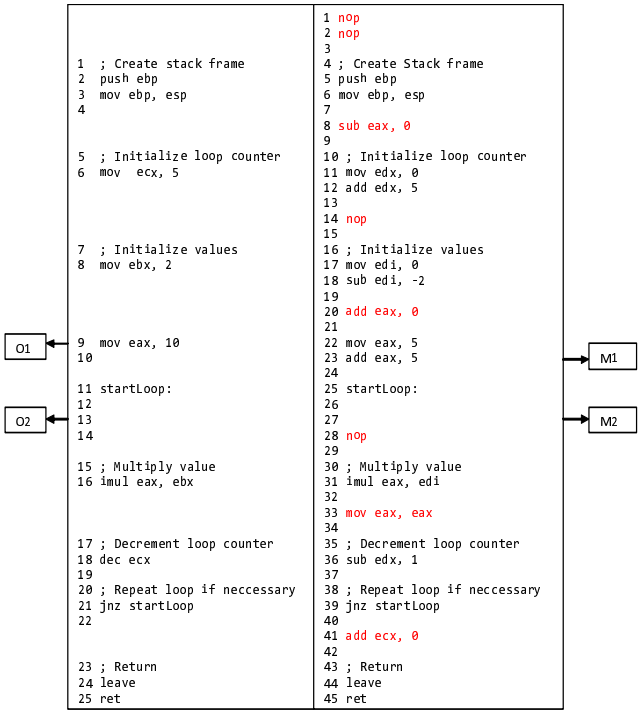

Figure 1. Metamorphic code sample

In the following a brief description of the overall identification process given. This is followed by a a more detailed explanation of the reliable identification of points of interest (POI) inside metamorphic malware. The last part explains the matching process.

4.1. Identification Process

The identification process of a suspicious binary program is performed in four consecutive steps.

1. Create disassembly
2. Analyze
3. Match points of interest
4. Calculate similarity score

The general concept of the value set analysis relies on the disassembly of the program analyzed. Even though this step might already be targeted by specially obfuscated programs, different methods exist to create reliable disassemblies even in those cases [18]. The value set analysis is then applied as a static analysis to approximate the possible values of each memory location for every instruction in the program. Since it is performed as a *static* analysis (instead of dynamic) all possible execution paths are analyzed. As a result randomization and time-dependent control-flows have no impact on the value sets. After all value sets have been computed they are matched to a reference list of value sets. This list contains the characteristics of a metamorphic specimen as described in the following. Based on the matching a similarity score is computed, which can be used for identification or classification.

4.2. Points of Interests

Metamorphic malware significantly changes its structure and appearance from infection to infection. Furthermore, many of the specimen are file-infecting and sometimes merge into the code of the host program. This results in two major challenges when trying to match possible characteristic value sets to those of a suspicious file. First, points inside the program have to be identified at which it is probable that value sets do not change from infection to infection. Those points will be called *points of interest* (POI) in the following. Secondly, POIs have to be found that can be used to distinguish values of the host program from those of the malware.

Prominent POI candidates are function calls, library calls, jumps, functions entries, a combination of those, or all instructions in the program. During function calls, certain parameters, like flags, can only be chosen from a specific range. This is especially true for external library functions that require specific constants for the functionality required, like the “protocol” parameter for the *socket()* function. At conditional jumps decisions are made depending on certain values. At function entries specific set-ups have to exist. Of course, considering all instructions assures that no characteristic value set is missed. Obviously, this requires the most complex analysis. Different POIs and their impact on the detection are evaluated in section 5.

When considering all instructions of a file-infecting malware, the value sets of instructions from the host program will also be included. In order to distinguish value sets derived from the host program from those from the malware,

we propose a refinement process. By using multiple samples with multiple host programs it is possible to extract the value sets relevant for the malware and to eliminate those of different host programs. This is performed by creating the intersection of value sets $\{V\}$ and data objects $\{C_i\}$ that can be found in all infected programs. Since a single value set can contain data objects from both host and malware, not only identical value set are used but also matches with at least a certain matching score. This is depicted in figure 2. The value set at a given POI contains multiple data objects. Two of them belong to the virus. One data object is the result of the host program's code. For two value sets of different files V_1 and V_2 the refinement score Λ is the ratio of data objects found in both value sets:

$$\Lambda := \frac{|V_1 \cup V_2|}{|V_1 \cap V_2|} \tag{4}$$

If two value sets of different files V_1 and V_2 match at least with the given threshold $\Lambda \geq \tau$, all their matching data objects are included in the further refinement process. The refinement is repeated with all dedicated files. The number of files needed for good refinements as well as suitable τ are discussed in the evaluation (section 5).

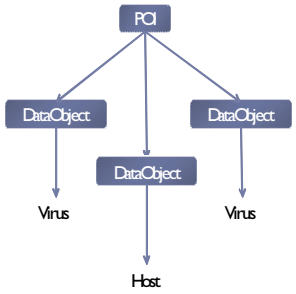

Figure 2. Refinement process

4.3. Matching

As described in 3, the value set analysis computes an over-approximation of all the values that each memory location can contain.

The actual matching process takes the values sets $\{V_s\}$ that characterize a metamorphic malware (source) and tries to find the *best matching* value sets $\{V_t\}$ in the considered sample (target). For this, the data objects of each $v_s \in \{V_s\}$ are matched to each $v_t \in \{V_t\}$ and a value set similarity is calculated. This is used to find the best matching combination, which then is used in the similarity rating for the whole file. Two value sets V_s and V_t are compared using the similarity of their data objects $C_{s,i}$ and $C_{t,i}$ of each set. Again, the *best matching* data objects are used to calculate the similarity rating of two value sets.

This double best matching approach is used to identify semantically corresponding parts inside completely morphed variants.

The computation of the similarity score of two data objects $C_{s,i}$ and $C_{t,i}$ is performed in three steps:

1. Intermediate score S' based on $\{s_{s,i}\} \times \{s_{t,j}\}$
2. Adjustment based on memory location $l_{s,i}$ and $l_{t,j}$
3. Normalization of the final score S_{final} to $[0, 1]$

In the first step the values $s_{s,i}$ and $s_{t,j}$ of two data objects are compared. Due to the refinement process, the source value sets $\{V_s\}$ that characterize the malware can be assumed to be more precise than the value sets obtained from the single target sample $\{V_t\}$. Thus, for matching two value sets $v_s \in \{V_s\}$, $v_t \in \{V_t\}$ with matching data objects $C_{s,i}$ and $C_{t,j}$:

$$s_{s,i} \subseteq s_{t,j} \tag{5}$$

Thus, if $s_{s,i} \supset s_{t,j}$ then $C_{t,j}$ does not match $C_{s,i}$ and a final similarity score of $S_{final} = 0$ is assigned. If not, an intermediate similarity S' is computed. When both $s_{s,i}$ and $s_{t,j}$ are exact sets of values and no over-approximations, an exact similarity is determined as:

$$S' := \frac{|s_{s,i} \cup s_{t,j}|}{|s_{s,i}|} \tag{6}$$

For many positions in a binary, the value set analysis leads to over-approximations. In this case, the intermediate score is set to $S' = 100\%$. Due to the assumption 5: $|s_{s,i}| \leq |s_{t,j}|$. The more they differ, the more unlikely is their similarity. The intermediate score is lowered by $\Delta_{cardinal}$ if $|s_{s,i}| < |s_{t,j}|$ and by Δ_{2card} if $|s_{s,i}| < 2 \cdot |s_{t,j}|$

In the second step, $l_{s,i}$ and $l_{t,j}$ are compared. If both are registers, the intermediate score is not modified. Registers can and are often exchanged in metamorphic malware. Therefore, registers have to be regarded as aliases for each other. In case one location is a register and the other is stack memory a degree of uncertainty is introduced. Thus, the similarity score is adjusted to $S' = S' - \Delta_{loc}$. Both locations on the stack indicate some similarity. Similar offsets indicate a higher similarity while a difference implies uncertainty. Therefore, the offsets are compared and the similarity score is lowered to $S' = S' - \Delta_{stack}$ if they differ.

In the third step the final score is normalized to $[0, 1]$ by setting $S_{final} = 0$ when $S' < 0$. Otherwise $S_{final} = S'$.

To sum it up, the score depends on the ratio of source values found in the target range. It is lowered for certain relations by Δ_{stack}, Δ_{loc}, $\Delta_{cardinal}$, and Δ_{2card}. The latter parameters are used for fine tuning and are determined in the first part of the following evaluation.

5. Evaluation

The evaluation has been performed in two phases. In the first phase, the impact of the different parameters in the matching and refinement process have investigated and an optimized parameter set has been determined. For this step, the Lexotan32 virus [8] has been used since it includes all major metamorphic techniques. The general applicability of this approach is evaluated by using the derived parameters for six metamorphic malware samples.

5.1. Parameter Derivation

The parameter set has been determined using 30 different files infected by Lexotan32 [8]. Five files are used for the refinement process and 25 are used for the evaluation. In addition, 25 programs from a standard Windows installation are used to check for false positives. Lexotan32 has been chosen because of including all major metamorphic techniques, like junk insertion, register exchange, code permutation, and instruction substitution. Lexotan32 is the evolution of different viruses [7]. Of the 30 samples uploaded to VirusTotal [9] the average detection ratio was 12.6% and none of the 40 virus scanner was able to detect all samples.

The matching score of two files depends on the score of the value sets V at the considered POIs. Similarly, the matching score of two value sets depends on the score of their data objects C. We have evaluated two weighing schemes for generating these scores: *Average scoring* and *Threshold scoring*. *Average scoring* computes the higher-level score by using the average of all lower-level scores. For files, this is the average score of all value sets. For value sets, this is the average score of the data objects. Average scoring includes very low scores that might indicate two objects not being similar. Therefore, *Threshold scoring* is evaluated as an alternative to compute the higher-level score solely based on those lower-level scores that reached a least a certain threshold. E.g. a threshold of 0.7 for file matching means that only those value sets are included in the average score that have at least a similarity score of 0.7. We have performed an extensive evaluation that investigates 48 set-ups with all combinations of the following settings using all instructions as POIs:

- Average scoring and Threshold scoring for files
- Average scoring and Threshold scoring for value sets
- For Threshold scoring: Thresholds of 0.7, 0.8, and 0.9
- Δ_{stack}, Δ_{loc}, $\Delta_{cardinal}$, and Δ_{2card} set to the same value: $\Delta = 0.1$, 0.2, and 0.3

We use a *differentiation threshold* Γ to classify whether a file is a Lexotan32 variant or not. For a file with a similarity score $S_{final} \geq \Gamma$, it is assumed to be infected. If $S_{final} < \Gamma$ it is assumed to be clean. Steady configurations have been found for $\Gamma = 0.8$ to 1.0 Figures 3 and 4 illustrate the numbers of false positives and negatives.. The results for $\Gamma = 0.9$ are similar. For $\Gamma \geq 0.8$, set-ups 34, 35, 36, 46, 47, and 48 have a perfect detection without any false positives or false negatives. All of these set-ups use Δ of 0.3 and Threshold scoring for both files as well as value sets.

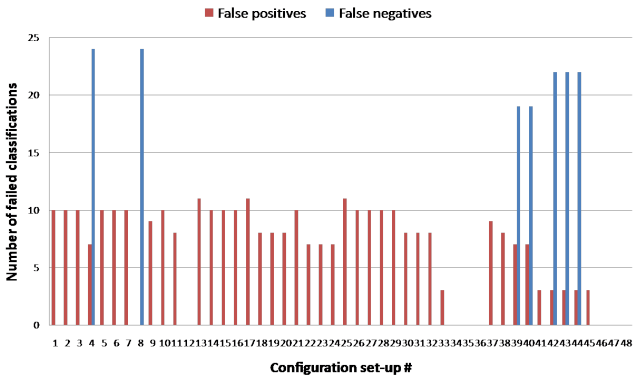

Figure 3. Detection errors for $\Gamma = 0.8$

Figure 4. Detection errors for $\Gamma = 1.0$

Since the relation of Δ values to the value set threshold scoring seems to be the important factor for perfect detection, the influence of Δ_{stack}, Δ_{loc}, $\Delta_{cardinal}$, and Δ_{2card} on the result has been examined. Each Δ was set to 0.3 while the others were set to 0.0. Two other tests were performed with all $\Delta = 0$ and all $\Delta = 0.3$ The detection of infected files were always perfect with a similarity score of 1.0, which yields 0% of false negatives. Thus, the use of Δ does influence the detection of infected files but only reduces false positives. Δ_{stack} and $\Delta_{cardinal}$ have not show an impact on the false positives. Δ_{loc} results in 8 false positives . Δ_{2card} reduces the false positives to 0 for $\Gamma \geq 0.75$. With all $\Delta = 0.3$ the use of lower Γ was even possible.

All in all, the use of high threshold scoring is required to

eliminate false negatives. All values sets are perfectly found in every infected files and $\Gamma = 1.0$ can be used. For the evaluation with malware, strict threshold values of 1.0 are chosen for value sets and data objects. With setting Δ_{2card} and Δ_{loc} to a level of 0.3, the similarity scores of clean files stays significantly below the given Γ. 84% of them have a similarity score of 0, anyway.

Besides the derivation of matching parameters, the refinement parameters have been determined. In the following, we will only present the results but not describe the evaluation itself. For including value sets $\{V\}$ in the refinement process, a *Refinement score* of $\tau \geq 0.7$ was found to be suitable. For this parameter, two refinement steps are enough to reach the final refined characterization. The initial analysis of all instruction POIs of the first file leads to 153 suitable value sets with 633 data objects. This was lowered to 4 value sets with 8 data objects after two refinement steps. These characteristics do not change in consecutive refinement steps.

5.2. Malware Evaluation

The main evaluation is based on the parameters presented in the previous part. They are used to evaluate the detection of six malware specimen with metamorphic code. In addition, we study the impact of the POI selection on the overall result.

The specimen used for evaluation are listed in the following table. Out of the six specimen, only W32/Evol is a full metamorphic. The other six are polymorphic, which means that their original code is encrypted and the decryptor is included and executed at the beginning of the virus code. These decryptors are metamorphic. Therefore, the evaluation is based on the detection of this metamorphic part. The VirusTotal column shows the detection result of 40 virus scanners used in VirusTotal [9]. The first number is the average detection rate, the second shows the number of virus scanners that detected every variant. None the viruses use metamorphism as sophisticated as found in Lexotan32, which has been used for the parameter evaluation in the first phase. This results in higher detection percentage compared to Lexotan32.

Name	ref.	Type	VirusTotal ⌀/perf.
W32/Evol	[5]	Full	74.1% / 23
W32/AOC	[4]	Decryptor	74.2% / 26
W32/BlackBat	[1]	Decryptor	70.5% / 21
W32/Bolzano	[2]	Decryptor	53.9% / 15
W32/Hatred	[6]	Decryptor	48.3% / 12
W32/Hezhi	[3]	Decryptor	68.0% / 25
W32/Lexotan32	[7]	Full	12.6% / 0

All of the presented malware specimen are file-infecting. Therefore 30 different files were infected with each of the malwares. Five of those are used for refinement. The left-over 25 are used for the actual evaluation. In addition, 25 programs from a standard Windows installation are used to check for false positives. The evaluation has been performed for all instruction POIs as well as for jumps, calls, and function entries.

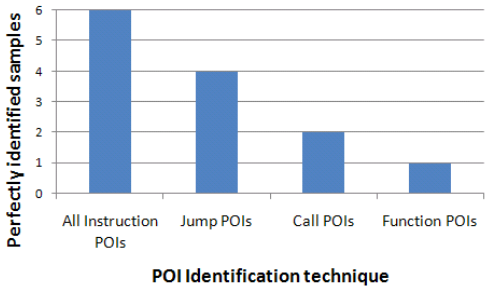

Figure 5. Number of perfect identifications

We define a *perfect identification* when both sets of each 25 infected and clean files are correctly identified. This means that neither false positives nor false negatives have occurred for any specimen.

Figure 5 shows the number of perfectly identified and differentiated specimen for the given POI types. Using all instruction POIs all six specimen are identified perfectly. Four out of the six malware were identified perfectly using jump POIs. Two specimen ended in having no value sets for identification for this POI after five refinement steps. Call and function entry POIs are less often available and lead to fewer numbers of value sets. After the refinement there were still two specimen that were identified perfectly with call POIs and even one specimen with perfect identification using function entry POIs.

The following table displays false positives and false negatives for each of the malware specimen.

Name	All Ins.	Jumps	Call	Func. Entry
W32/Evol	√	√	√	4% / 0
W32/AOC	√	4% / 0	-	-
W32/BlackBat	√	√	√	√
W32/Bolzano	√	√	-	-
W32/Hatred	√	√	-	-
W32/Hezhi	√	-	-	-

"√" indicates perfect identification, i.e. 0% false positives and 0% false negatives. "-" means that no value sets are available for characterizing the malware after five refinement steps. As can be seen by the table, there is either a perfect identification or no value sets available with two exceptions. The use of function entry POIs leads to one false positive for W32/Evol. The number of false negatives is still zero, which means that all infected files have been identified correctly. The same situation exists for the jump POIs of W32/AOC. All in all, in cases in which

characterization data exists, all infected files have had a similarity score of 1.0, which means perfect detection. The score of uninfected files depends on the POI selection with all instructions being the most reliable strategy.

6. Conclusions and Future Work

We have presented an approach to detect metamorphic malware by using characteristic values that are used in all variants of given specimen. These characteristic values describe parts of the behavior of the malware. We extract the values with the use of a value set analysis. In order to get specific characteristics, a refinements process is applied that uses the intersection of values extracted from different variants. For file infecting viruses, this step allows us to separate the value sets of the host program from those of the malware.

Two different matching schemes and specific parameters are used to achieve the best detection. These have been evaluated and their influence on the matching performance determined. With this set of parameters together with using all instruction POIs, we were able to perfectly identify all seven considered metamorphic malware specimen without any false positives or false negatives.

The results for other POIs are less perfect because not enough characteristic values have been found at those. One reason is the strict matching that has been used to extract those. The combination of POIs to might lead to characteristics while lowering the analysis effort compared to considering all instructions. A larger number of malware and clean samples might be beneficial for fine-tuning the parameters. Furthermore, the approach could also be useful for general identification of malware instead of metamorphic only. An evaluation of this will be future work.

7. Acknowledgements

The authors would like to thank the anonymous reviewers of this paper for discussions and comments. We are also thankful for the people who supported us and gave valuable suggestions.

References

[1] Blackbat virus (last visit (lv): Jun. 2009) http://www.rohitab.com/sourcecode/blackbat.html.

[2] Symantec security response, win32/bolzano (lv: Jun. 2009) http://www.symantec.com/security_response/writeup.jsp?docid=2000-121515-4146-99&tabid=2.

[3] Symantec security response, win32/hezhi (lv: Jun. 2009) http://www.symantec.com/security_response/writeup.jsp?docid=2003-032615-3916-99&tabid=2.

[4] Virus encyclopedia, win32/aoc (last visit: Jun. 2009) http://www.viruslist.com/en/viruses/encyclopedia?virusid=20299.

[5] Virus encyclopedia, win32/evol (last visit: Jun. 2009) http://www.viruslist.com/en/viruses/encyclopedia?virusid=20535.

[6] Virus encyclopedia, win32/hatred (last visit: Jun. 2009) http://www.viruslist.com/en/viruses/encyclopedia?virusid=20608.

[7] The molecular virology of lexotan32: Metamorphism illustrated. http://www.openrce.org/articles/full_view/29, last visit: Apr. 2009.

[8] 29a #6, virus magazine. http://vx.org.ua/29a/29A-6.html, last visit: Jun. 2009.

[9] Virustotal - free online virus and malware scan. http://www.virustotal.com/, last visit: Jun. 2009.

[10] Tool interface standard (tis) executable and linking format (elf) specification. *TIS Committee*, May 1995.

[11] G. Balakrishnan. Wysinwyx: What you see is not what you execute. *Ph.D. dissertation and Tech. Rep. TR-1603, Computer Sciences Department, University of Wisconsin, Madison, WI*, Aug. 2007.

[12] D. Chess and S. White. An undetectable computer virus. *In Proc. of the 2000 Virus Bulletin Conference (VB2000)*, 2000.

[13] S. Coorp. Symantec internet security threat report. *Whitepaper*, Volume XIII, Apr. 2008.

[14] M. M. D. Bruschi, L. Martignoni. Detecting self-mutating malware using control-flow graph matching. *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), IEEE Computer Society*, 2006.

[15] M. M. D. Bruschi, L. Martignoni. Using code normalization for fighting self-mutating malware. *In Proceedings of International Symposium on Secure Software Engineering, Washington, DC, USA*, 2006.

[16] M. C. et al. Malware normalization. *Technical Report 1539, University of Wisconsin, Madison, Wisconsin, USA*, Nov. 2005.

[17] H. Flake. Structural comparison of executable objects. *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), IEEE Computer Society*, 2004.

[18] A. Kapoor. An approach towards disassembly of malicious binary executables. *Masters thesis, The Center for Advanced Computer Studies, University of Louisiana at Lafayette*, Nov. 2004.

[19] Microsoft. Microsoft portable executable and common object file format specification. http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx, January 2009.

[20] R. L. Rivest. The md5 message-digest algorithm (rfc 1321). http://www.ietf.org/rfc/rfc1321.txt?number=1321.

[21] P. Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.

[22] Q. Zhang and D. S. Reeves. Metaaware identifying metamorphic malware. *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, 2007.