# A Statistical Approach for Discovering Critical Malicious Patterns in Malware Families

Vida Ghanaei, Costas S. Iliopoulos, and Richard E. Overill

Department of Informatics, King's College London
Email: {vida.ghanaei, c.iliopoulos, richard.overill}@kcl.ac.uk

***Abstract*—In this paper, we present carefully selected critical malicious patterns, which are in common among malware variants in the same malware family, but not other malware families, using statistical information processing. The analysed critical malicious patterns can be an effective training dataset, towards classification of known and unknown malware variants. We present malware variants as a set of hashes, which represent the constituent basic blocks of the malware Control Flow Graph, and classify them into their corresponding malware family. By computing the Distribution Frequency for each basic block residing in all the malware families, the importance of being a possible representative to become a critical malicious pattern for a specific malware family is measured. This value is carefully computed by considering the population of each malware family.**

***Keywords–Malware; Malicious Patterns; Malicious Shared Code; Classification; Control Flow Graph; Numerical Statistics.***

## I. INTRODUCTION

Malware is considered a major computer security problem as it can attach to other computer programs and infect them. Infection is defined as unwanted modification to other program to include a possibly evolved, version of itself [1]. Based on McAfee threats report [2], more than 30 million new malware variants, and over 250 million in total, are recorded in the first quarter of year 2014. Malware spread itself by most common digital methods such as e-mail and instant message applications, and through social engineering techniques. Other means of malware spread methods are World Wide Web (WWW), network-shared file systems, peer to peer file sharing networks, removable media, Internet Relay Chat (IRC), Blootooth or wireless local area networks [3].

In malware research, data collection is not an issue any more. It is easily achievable by setting up configured honeypot servers in the laboratory environment. But, the flow of malicious software variants reaching our networks and computers is so enormous that makes it impossible to process them exclusively. Therefore, it is essential, to identify malicious patterns which appear in malware variants, and to obtain structural understanding of malicious patterns, in order to analyse, classify, and detect malware. However, collection of huge amount of malware variants, which have embodied different obfuscation techniques, and have mutated in various polymorphic and metamorphic forms, demands automated techniques to identify, and present the malicious patterns. While malware variants belonging to the same malware family[1] share a certain amount of malicious code, identification of critical shared code provides knowledge towards classification and detection of unknown malware variants.

[1]Defined in Section III.

In this paper, we present an effective statistical approach to identify, and to render critical malicious patterns in malware families, which are essential elements towards automated classification of known and unknown malware in large amount. We rely on the shared code among different malware variants, which potentially occur in one specific malware family, with considering its possibility of occurring in other malware families, to identify the most critical malicious patterns in every malware family. In this paper, the shared code is studied at basic block level of the control flow related code, and we are able to present the most critical malicious patterns for every malware family. By critical malicious pattern, we mean the most frequent basic blocks, which are present at most in one specific malware family, and comparatively less in other malware families. Also, the shared code among different malware families are not interesting for classification purpose, as certain functions are in-common among all the malicious software variants, and even can be in-common with non-malicious software.

We introduce a novel formalisation methodology which automates the identification of critical malicious patterns for each malware family. It is defined as a statistical approach, which computes the Frequency Distribution Ratio, for each constituent basic block of a malware variant within each malware family. This value is penalised statistically for occurring in other malware families. To our knowledge, our approach in compare to related works, is more consistent as we encounter the distribution frequency of certain basic blocks in each malware family, as well as in between different malware families, to identify critical malicious patterns. Not considering the ratio of the frequency of each basic block in its associated malware family, results in inaccurate identification of critical malicious patterns, which is discussed in details, in Section VIII.

In Section II, the background and related work on different malicious pattern matching techniques are reviewed. In Section III, the notations and definition are given. In Section IV, details of the dataset used in experiments, are presented. In Section V, the formalisation of our approach is defined. In Section VI, an overview on the shared code concept in malware, and how we bypass the obfuscations applied to shared code, is explained. In Section VII, the methodology and implementation process is described. In Section VIII, experimental results of our approach is discussed and compared to related works. Finally in Section IX, our methodology, its results, and the future work are concluded.

## II. BACKGROUND AND RELATED WORK

Different features of malicious software, and pattern matching techniques are studied to classify enormous amount of

 ISBN: 978-1-61208-393-3 

malware variants getting submitted to honeypots. The n-gram, defined in section III., computation as a pattern matching techniques, and its application in malware analysis, was first introduced by G. J. Tesauro et al [4], to identify boot sector viruses automatically by applying artificial neural networks. J. Z. Kolter et al[5], computed Information Gain (IG) for each n-gram to select the most relevant n-grams, to heuristically identify and classify malware. They selected top 500 n-grams, and applied different machine learning algorithms to detect and classify malware variants and identified boosted J48 algorithm produces the best detector. Although they showed good detection with areas under the ROC curve around 0.9, they did not consider metamorphic obfuscations and polymorphism.

S. Cesare et al [6], presented a static approach to detect polymorphic malware based on control flow graph[1] classification. They unpacked[1] the polymorphic malware using application level emulation, disassembled the unpacked samples, translated the disassembly into intermediate language, reconstructed and normalised the control flow graph for each procedure based on the intermediate language. Subgraphs of size *K*, and n-grams extracted from the strings representing the graphs, are the two features used to pre-filter potential malware, and used to construct a feature vector. Distance metrics are used to measure similarity between two feature vectors. The presented results shows considerable collisions, and false positives using subgraphs of size *K* compare to using n-grams vector features.

BinDiff[2], which is an add-on plug-in for IDAPro[3], relies on heuristics to generate matches between two malware variants. It generates a signature for every function of the malware executable based on its abstract structure, ignoring the assembly code generated by IDAPro. The signature generated depends on the structure of the normalised flow graph of the function, and consists of number of basic blocks, number of edges, and number of calls to sub-functions. Two functions match, if a signature occurs in both only once. If a match identified, all the calls-to relations between the matched functions is checked for possible matches of all the subset functions, until no further matches found. BinDiff is a pairwise matching tool and generates lots of false matches as it relies on further matches on big portion of the code, rather than the most critical code.

C. Miles et al [7], presented a recent artefact, VirusBattle, which is a malware analysis web-service. VirusBattle reason about malware variants in different levels of abstraction including the code, the statically analysed shared semantics, referred as juice [8], among different variants, and sequence of events a malware takes during execution time to map the similarities and interrelationships. Juice, transforms code semantics computed over an x86 disassembly, by generalising the register names, literal constants, and computing the algebraic constrains between the numerical variables. Therefore, semantically similar code fragments can be identified by comparing their hash values. VirusBattle provides automated PE unpacking web-service as well, which is a generic unpacker[4], and publicly available web-service. In this paper, we use the unpacker provided by the VirusBattle SDK to unpack the malicious samples, as well as the juice which is the generalised presentation of the semantics to avoid code obfuscations.

[2]Zynamics and Google, BinDiff. Available: http://www.zynamics.com/bindiff.html

[3]I. Guilfanov, IDAPro, An Advanced Interactive Multi-processor Disassembler, Data Rescue. Available: http://www.datarescue.com

[4]V. Notani and A. Lakhotia, VirusBattle SDK-Unpacker. Available: https://bitbucket.org/srl/virusbattle-sdk/wiki/Home

## III. Definitions and Notations

*a) Malware Taxonomy:* Malware is a general term for malicious software, which refers to virus, worm, root-kit, trojan-horse, dialer, spyware, and key-logger. It is defined based on its mean of distribution, and its dependency on the host to infect.

*b) Malware Family:* Malware variants which are the result of the mutation of each other, and share considerable amount of critical code with one another, are considered to belong to the same malware family. The already known malware variants belonging to the same malware family, can be classified by signature based anti-virus scanners. Variants in the same malware family are meant to show similar behaviour, to target similar files, and to spread the same way. Therefore, each malware family, denoted $f$, is a set of malware variants.

*c) Packed Malware:* Malware which contains encryption routine, compression, or both, is referred to as packed malware. Unpacked malware is the malicious code, without encryption or compression.

*d) Control Flow Graph:* Control Flow Graph (CFG), is a directed graph, is denoted *G = (V, E)*, if $u, v \in V$ be the nodes of the graph, a possible flow of control, from *u* to *v* is represented by $e \in E : u \rightarrow v$. In a CFG, every node is a representation of a basic block, denoted $v$, and the edge is a path between these nodes. A basic block is defined as a sequence of instructions without any jumps or jump targets in between the instructions [9].

A basic block always runs before any other instructions in later positions of the CFG, which means no other instruction runs between two instructions in the same sequence. The directed edge between two basic blocks expresses the jump command in the control flow, which is caused by Control Transfer Instructions (CTI) such as call, conditional jump, and unconditional jump instructions [10].

*e) N-gram Frequency Distribution:* N-gram is a contiguous sequence of *n* items from a given sequence such as assembly statement raw bytes, opcodes and etc. N-gram frequency distribution, is a well-known approach for extracting features from malicious software to develop training dataset for classification purpose [5]. In this article, the hash value computed for every basic block is treated as n-gram sequence.

## IV. Dataset

Our dataset consists of 777 distinguished malware variants, which are spread over 23 different malware families, as shown in Table I., and it contains total of $1,116,798$ basic blocks. Malware families are structured disjoint, to avoid inaccurate computation of critical malicious patterns. If the same malware be a member of different malware families, its constituent basic blocks will be counted towards all the involved malware families and cause false matches. Also, in the classification process of malware variants, a new sample is to be placed in one malware family based on the similarity measurement of its shared code with that malware family as oppose to other malware families. However, each malware family is treated as a multiset of basic blocks. In other words, each basic block

 ISBN: 978-1-61208-393-3 

can occur multiple times in the same malware family, and in other malware families as well. Each basic block is indexed to its associated malware variant to produce more informative outcome.

Malware samples are collected from the VirusSign[5] free on-line service, and the dataset presented in a recent study [11]. All the malware samples existing in the database, are unpacked by the VirusBattle SDK unpacking service[6].

TABLE I. MALWARE FAMILIES EXISTING IN DATASET

| Malware Family | No of Variants | No of Basic Blocks |
|---|---|---|
| Agent | 17 | 12316 |
| Agobot | 17 | 101782 |
| ATRAPS | 9 | 16064 |
| Bancos | 13 | 33831 |
| Cosmu | 8 | 1133 |
| Crypt | 63 | 239821 |
| Dldr | 12 | 82929 |
| Downloader | 9 | 5769 |
| Dropper | 63 | 75082 |
| Fareit | 146 | 64819 |
| Gobot | 14 | 28451 |
| IRCbot | 17 | 11405 |
| Klez | 22 | 69721 |
| Kryptik | 97 | 49696 |
| MyDoom | 40 | 67169 |
| Poebot | 18 | 9285 |
| Sality | 54 | 26379 |
| Spy | 26 | 38511 |
| Symmi | 18 | 23600 |
| ULPM | 65 | 94165 |
| Unruy | 18 | 2407 |
| Vanbot | 16 | 1713 |
| Virut | 27 | 15750 |

## V. FORMALISATION

Considering each malware family as a set of malware variants, and each malware variant be a set of multiple basic blocks. The malware families are disjoint, in other words each malware variant can only be a member of one malware family. However, $v_i$, is the $i$th distinct member of a $v$, which can occur in any malware family and multiple times.

Let $f_k$ be the $k$th distinct member of $f$, and $\tau_{i,k}$ be the number of occurrences of $v_i$ in $f_k$, and $\tau_k = \sum_i \tau_{i,k}$. Therefore, Term Frequency Ratio ($TFR$) for $v_i$ occurring in $f_k$ is defined as shown in Equation 1.

$$TFR_{i,k} = \frac{\tau_{i,k}}{\tau_k} \tag{1}$$

$TFR$ is computed for all the $v_i$, which exist in every malware family, individually. $TFR_{i,k}$ indicates the ratio of how frequently $v_i$ has occurred in $f_k$. Every $v$ is indexed to its associated malware variant. The frequency ratio of each $v_i$ is considered, rather than the frequency of each $v_i$, as the number of malware variants in each malware family is varying and consequently the $\tau_k$. Therefore, it is essential to encounter the the population of malware families in compare to each other, to obtain the correct frequency for each $v_i$.

Here, we compute the importance of each $v_i$, as a possible representative for the critical malicious pattern of each malware family. In order to do so, its TFR value is penalised by subtracting a quantity $\alpha_{i,k}$. $\alpha_{i,k}$ is the sum of $TFR_{i,j}$ of all the occurrences of $v_i$ in all the malware families, $f_j$, where $j \neq k$; except $f_k$. It is computed for every malware family in relation to its population. Therefore $\alpha_{i,k}$ is defined as shown in Equation 2.

$$\alpha_{i,k} = \sum_{j \neq k} TFR_{i,j} \tag{2}$$

Therefore, Term Frequency Distribution ($TFD$) for $v_i$, computes the frequency ratio of $v_i$ in malware family $f_k$, in relation to its distribution in the other malware families, which is $\alpha_{i,k}$, and it is defined as shown in Equation 3.

$$TFD_{i,k} = TFR_{i,k} - \alpha_{i,k} \tag{3}$$

$TFD_{i,k}$ indicates how critical $v_i$ is as a malicious pattern for $f_k$, as $v_i$ is penalised for every time it occurs in $f_j$. The penalising as described, is computed by subtracting the sum of all the corresponding $TFR_{i,k}$ of $v_i$ occurrences in $f_j$. This sum, $\alpha_{i,k}$, indicates how common $v_i$ is in other malware families, $f_j$, rather than $f_k$. Accordingly, $TFD_{i,k}$ shows how specific $v_i$, is to malware family $f_k$, rather than other malware families, $f_j$.

## VI. SHARED BASIC BLOCKS

Malware variants belonging to the same malware family, share certain amount of code which relates them. The syntax of the shared code in every variant varies every time malware mutates, which is due to applied obfuscation techniques to avoid detection by anti-virus scanners. Different syntax presentation for the same semantics, causes many mismatches. To overcome incorrect matches that result from these syntax changes, we use the VirusBattle SDK on-line platform to generate the same syntax for blocks of code, which carry the same semantics. Therefore, malware samples are unpacked using VirusBattle SDK unpacker, and the juice, which is the generalised presentation of the semantics, is generated for every sample using the on-line service provided by the same platform. Here, semantics refers to the semantics of instructions in the basic block, and is developed based on the original disassembled code after unpacking. We use this service as it has shown good results and serves our needs well.

## VII. METHODOLOGY AND IMPLEMENTATION

According to previous studies by C. Miles et al. [7], and Zynamics[2], the CFG has shown the best results for malware similarity measurement purposes. Therefore, we chose to identify shared code among different malware variants by looking into their CFG, and at the basic block level. Basic blocks are preferred to instructions, as instructions become meaningless without contextually relating to other instructions, semantic-wise, and functions can contain many basic blocks and produce incorrect matches, such as the work presented in Zynamics. For every malware sample, the juice corresponding to each

[5]VirusSign. Available: https://www.virustotal.com

[6]V. Notani and A. Lakhotia, VirusBattle SDK-Unpacker, 2014. Available: https://bitbucket.org/srl/virusbattle-sdk/wiki/Home

 ISBN: 978-1-61208-393-3 

of its basic blocks is hashed and stored in a text file. Thus, each malware is presented as a multiset of hash values, which resembles its constituent basic blocks. Hashing is applied to accelerate the matching process.

Malware samples are initially scanned, and labeled by Avira[7] anti-virus scanner, and through the Open Malware[8] on-line service, to be pre-classified into their related malware families, and stored accordingly. Each malware sample can be a member of one malware family only, and a warning message is flagged if duplications found. Duplication of malware variants is avoided by computing their hash value. Therefore, each malware family consists of set of malware variants in which, each malware variant is a multiset of $v_i$. Duplicate malware variants are restricted, as the aim of this paper is to identify critical malicious patterns, which can be used as the training dataset for malware classification purpose, and allowing multiple copies of the same malware variant, results in wrong frequency ratio of the shared basic blocks.

The critical malicious patterns for each malware family is described as a list of basic blocks, which occur in malware family $f_k$ the most, and are least likely to occur in other malware families, and computed by TFD for each $v_i \in f_k$. In other words, the importance of basic block $v_i$ is lowered by its occurrence in different malware families, as it is not implicit to a specific malware family, therefore not the best candidate for being categorised as the critical malicious pattern. However, due to the fact that the amount of shared code between different malware families is considerably high, not many distinct basic blocks to one malware family, are ranked as its critical malicious pattern. Nonetheless, our formalisation is defined by considering these fundamentals, by studying the distribution frequency of each basic block in each malware family, as well as in between all the malware families. Hence, the inclusion of basic blocks, which carry less specific functionality to a malware family, as its critical malicious pattern, is avoided. Therefore, the main fundamentals in defining the formalisation encountered are expressed as, each malware family consists of different number of malware variants, as well as each malware variant consists of different number of basic blocks depending on the amount of code involved in its CFG. Also, basic block $v_i$ may occur in different malware families. Thus, without considering the ratio of the frequency of each basic block, rather than its frequency in each malware family, and its distribution frequency in between other malware families, it is impossible to identify the correct critical malicious patterns for each malware family. According to our testing and experiments, these criteria are necessary to be encountered to define an effective formalisation in order to identify Critical malicious patterns for each malware family.

## VIII. Experimental Results and Discussion

The developed command line interface provides the ability to add or delete a malware family, to list already existing malware families, and to compute and update the TFD value for every basic block stored in the database. Also, by making different queries, it is possible to obtain the TFD for each basic block in its associated malware, in its malware family, and in compare to other malware families. It is possible to query a single basic block, and observe in which of the malware families it has occurred, and obtain its TFD value corresponding to each of the malware families. As shown in Listing 1., query for one basic block is made, which shows it is occurred in three malware families, Fareit family, Dropper Family, and the Klez Family. It also displays the number of times it has occurred in each of the malware families, and the total number of malware variants, which is 22. However, in Listing 1. we only show the first two malware variants, due to limited space.

The same way, we can query for one specific malware, and list all of its basic blocks. For each basic block, it shows list of the malware families in which the basic block occurs, the count of times it occurs in each malware family, and the count of malware variants which contain that basic block in one malware family. These queries are developed to understand the distribution of each basic block in its associated malware family, as well as the whole database. Observing these information provides an effective understanding on the shared code among different malware families, which is essential for the classification purpose.

The most critical malicious pattern for each malware family, is supposed to be the constituent basic blocks of that malware family with the highest TFD values. As shown in Listing 2., the top 10 patterns for the Sality family is queried. Sality family is chosen randomly as a sample, and all of the malware families, as shown in Table I., are included in the database, and can be queried. The result displays, the basic blocks based on their TFD value in descending order. Therefore, the result of this query presents a sorted list, in which for each basic block, total number of its occurrences, its TFR value, and its TFD value are retrieved. Furthermore, another query lists the occurrences of $v_i$, in the asked malware family, as well as all the other malware families, $f_j$. This query provides a good understanding on the shared basic blocks between $f_k$, and the other malware families, $f_j$. These shared basic blocks are considered to be strong candidates for the critical malicious pattern of the malware family, as they carry high TFD value. Both of the queries can be made to display any number of the basic blocks, in descending order based on the TFD value, as long as the value is equal or less than $\sum_i v_i$.

The details of the occurrences of $v_i$, in other malware families, $f_j$, as shown in Listing 3. for the first 5 basic blocks of the Sality family, with the highest TFD value, is given. As listed, the 3rd basic block has occurred in other malware families as well, which reveal the importance of the same basic block in other malware families. This basic block has occurred 344 times in Sality family, and 186 times in 11 other malware families, $f_j$. However, as explained before the frequency distribution ratio of a basic block, has the main impact on how critical it can be for a malware family, in terms of critical pattern identification. Further more, the hash value of the basic blocks, can be traced back into its juice, and the actual code for further analysis, as shown in Table II. In this paper, retrieving the actual code associated to each of the hash values, is traced manually as our aim is to identify the malicious patterns. However, it is a straight forward task to automate this process, as the name of the malware families, malware names, and the basic blocks identifier are the computed hash value, for each.

As mentioned in Section VII., different fundamentals regarding the shared code impacts the formalisation of malicious

[7] A.O.G. and Co., Avira Antivirus. Available: http://www.avira.com
[8] G.T.I.S. Center, Open Malware, Offensive Computing.

 ISBN: 978-1-61208-393-3 

```
DTF> query bb 5707c1cf0477658dd909009e519b179c426a3308;
Query basic block 5707c1cf0477658dd909009e519b179c426a3308
3 families:
Fareit: DTF:0.000570275505, TFR:0.000694240886, Count:45
Dropper: DTF:-0.000685018552, TFR:0.000066593857, Count:5
Klez: DTF:-0.00070346322, TFR:0.000057371524, Count:4
22 malwares:
ca7811eee67d6b340c07ff0dbd285193.txt (Klez)
3396ff4b55e6d1e7e133a6df027d55bb.txt (Fareit)
```

Listing 1. QUERY FOR ONE SPECIFIC BASIC BLOCK IN ALL THE DATABASE

```
DTF> query top 10 Sality;
1. BasicBlock{Hash=d99605bb279fd2e455e2c8a10643e1074094e929, count: 990, TFR: 0.037529853292, TFD: 0.037529853292}
2. BasicBlock{Hash=88629f540b9724221a6c0b43a7029276decb2f0c, count: 232, TFR: 0.008794874711, TFD: 0.008794874711}
3. BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 344, TFR: 0.013040676296, TFD: 0.008393762452}
4. BasicBlock{Hash=ebd4c8c2640ad80791d44ed1a7f818e1c08313ab, count: 218, TFR: 0.008264149513, TFD: 0.008264149513}
5. BasicBlock{Hash=517929ec5e483f5e6865568da0d13b10aa5a89fb, count: 212, TFR: 0.008036695857, TFD: 0.008036695857}
6. BasicBlock{Hash=e8e12af90c2895296d017e132c400843b586105c, count: 175, TFR: 0.006634064976, TFD: 0.006634064976}
7. BasicBlock{Hash=e7857a3b9d0e5364b70bc144cfb39dd900b80091, count: 133, TFR: 0.005041889382, TFD: 0.005041889382}
8. BasicBlock{Hash=222c5f3c8b5f052b5080b28d8735a92e446e50c3, count: 69, TFR: 0.002615717048, TFD: 0.002615717048}
9. BasicBlock{Hash=d60a105290a010fa52fde922d76c26181f4a2982, count: 77, TFR: 0.002918988589, TFD: 0.002379954337}
10. BasicBlock{Hash=eb159e43755db866112381d9d7ce4513269ba23d, count: 58, TFR: 0.002198718678, TFD: 0.002198718678}
```

Listing 2. SALITY FAMILY, 10 BASIC BLOCKS WITH THE HIGHEST TFD VALUE

```
DTF> query top 5 Sality more;
1. BasicBlock{Hash=d99605bb279fd2e455e2c8a10643e1074094e929, count: 990, TFR: 0.037529853292, TFD: 0.037529853292}
2. BasicBlock{Hash=88629f540b9724221a6c0b43a7029276decb2f0c, count: 232, TFR: 0.008794874711, TFD: 0.008794874711}
3. BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 344, TFR: 0.013040676296, TFD: 0.008393762452}
        1. Family{name: IRCbot} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 3, TFR: 0.000263042525, TFD: -0.017161505089}
        2. Family{name: Agent} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 1, TFR: 0.000081195193, TFD: -0.017525199752}
        3. Family{name: Dropper} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 10, TFR: 0.000133187715, TFD: -0.017421214709}
        4. Family{name: Virut} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 8, TFR: 0.000507936508, TFD: -0.016671717123}
        5. Family{name: ATRAPS} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 6, TFR: 0.000357824427, TFD: -0.016971941284}
        6. Family{name: ULPM} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 54, TFR: 0.000573461477, TFD: -0.016540667185}
        7. Family{name: Unruy} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 1, TFR: 0.000010857174, TFD: -0.017665875791}
        8. Family{name: Fareit} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 19, TFR: 0.00029312393, TFD: -0.01710134228}
        9. Family{name: Gobot} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 58, TFR: 0.002038592668, TFD: -0.013610404803}
        10. Family{name: Crypt} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 15, TFR: 0.000062546649, TFD: -0.01756249684}
        11. Family{name: Bancos} BasicBlock{Hash=3b48fe121b3cd7d3e3ec568778ae35a55df5a87e, count: 11, TFR: 0.000325145577, TFD: -0.017037298986}
4. BasicBlock{Hash=ebd4c8c2640ad80791d44ed1a7f818e1c08313ab, count: 218, TFR: 0.008264149513, TFD: 0.008264149513}
5. BasicBlock{Hash=517929ec5e483f5e6865568da0d13b10aa5a89fb, count: 212, TFR: 0.008036695857, TFD: 0.008036695857}
```

Listing 3. SALITY FAMILY, DETAILS ON THE OCCURRENCES IN THE OTHER MALWARE FAMILIES, OF THE HIGHEST 5 TFD VALUES

TABLE II. SALITY FAMILY, *v*:3b48fe121b3cd7d3e3ec568778ae35a55df5a87e SEMANTIC EQUIVALENT

| Code | $nop$<br>$mov(eax, none(eax-4))$<br>$nop$ |
|---|---|
| Semantics | $eax = none(eax-4)$ |
| Juice | $A = none(A-B)$ |
| Hash value of Juice | 3b48fe121b3cd7d3e3ec568778ae35a55df5a87e |

pattern identification. As part of our experiments, we have implemented the IG algorithm, as applied by I. Santos et al.[12], to learn about the shared code, and the main flaw of this algorithm in the context of shared code which we observed, is not considering the significant of the frequency ratio in between the malware families. However, this flaw may have not affected the work presented by I. Santos et al.[12], as they studied the opcode sequences, and not the basic blocks. In this paper, we aim to identify the critical malicious patterns as basic blocks, as CFG of every malware variant carries information regarding its functionality. Therefore, it provide us with informative pieces of critical malicious codes for each malware family.

The main drawback of our approach is the size of the database. Obviously, more number of malware variants, and malware families, will provide more accurate information regarding the critical malicious patterns. Nonetheless, in this paper we are proofing our proposed concept. However, malicious software includes more information than the shared code, such as strings, the information contained in the import-export tables of the malware, the file type, etc. We contend that our proposed approach is simple, and yet accurate, and effective. The presented approach can be applied to other features of the malicious software, by extracting the feature of interest, computing its hash value, and applying the TFD to obtain the pattern for each malware family. Thus, our presented methodology can be used to retrieve critical information about each malware family, which is essential to understand malicious software. Also, generating an effective general pattern for the numerous variants of each malware family.

The evaluation of our approach is simple, as the formalisation is based on the statistics. The counts and the implementation outcomes are checked manually, and they are accurate. The choices of the presented malware family, and the basic block are in principle arbitrary. Due to the automation the flowchart of the experimental process, is essentially trivial.

 ISBN: 978-1-61208-393-3 

## IX. Conclusion and Future Work

In this paper, we presented a statistical approach to generate, and discover critical malicious patterns in malware families. We presented the critical malicious pattern for each malware family, as a list of basic blocks, which represent the most frequent shared code among each malware family by considering its occurrences in other malware families. Here, we present the impact of occurrence of a basic block in different malware families, on its potential to be a candidate of critical malicious pattern. Our developed framework to extract the most frequent shared code, can be applied to any other feature of malware for further analysis. The generated critical malicious patterns, are the initiative for our future work, towards the classification, and detection of known and unknown malware. Also, in the future work, we consider to extend the size of the database.

## Acknowledgment

The authors would like to thank Golnaz Badkoubeh for her insightful comments on the formalisation method, to thank Ehsun Behravesh, Aristide Fattori, Danut Niculae, and Michal Sroka for their generous help in the implementation of the framework, and to thank Arun Lakhotia and Vivek Notani for their support and willing to provide the VirusBattle SDK platform. Also, to thank the anonymous reviewers for the helpful suggestions.

## References

[1] F. Cohen, “Computer Viruses: Theory and Experiments,” Computers & Security 6, 1987, pp. 22–35.

[2] C. Beek et al., “McAfee Labs Threats Report,” McAfee Labs Threats Report, Tech. Rep., Aug. 2014.

[3] O. f. E. Co-operation and Development, Computer Viruses and Other Malicious Software: A Threat to the Internet Economy. OECD, Mar. 2009.

[4] G. J. Tesauro, J. O. Kephart, and G. B. Sorkin, “Neural networks for computer virus recognition,” IEEE Expert, vol. 11, no. 4, Aug. 1996, pp. 5–6.

[5] J. Z. Kolter and M. A. Maloof, “Learning to Detect and Classify Malicious Executables in the Wild,” J. Mach. Learn. Res., vol. 7, Dec. 2006, pp. 2721–2744.

[6] S. Cesare and Y. Xiang, “Malware Variant Detection Using Similarity Search over Sets of Control Flow Graphs,” in Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, Nov. 2011, pp. 181–189.

[7] C. Miles, A. Lakhotia, C. LeDoux, A. Newsom, and V. Notani, “VirusBattle: State-of-the-art malware analysis for better cyber threat intelligence,” in Resilient Control Systems (ISRCS), 2014 7th International Symposium on. IEEE, 2014, pp. 1–6.

[8] A. Lakhotia, M. D. Preda, and R. Giacobazzi, “Fast location of similar code fragments using semantic ’juice’,” in Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop. ACM, 2013, p. 5.

[9] A. Moser, C. Kruegel, and E. Kirda, “Limits of Static Analysis for Malware Detection,” in Computer Security Applications Conference, 2007. ACSAC, Twenty-Third Annual, Dec. 2007, pp. 421–430.

[10] G. Vigna, “Static Disassembly and Code Analysis,” in Malware Detection, ser. Advances in Information Security, M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, Eds. Springer US, 2007, vol. 27, pp. 19–41.

[11] S. L. Blond, A. Uritesc, C. Gilbert, Z. L. Chua, P. Saxena, and E. Kirda, “A Look at Targeted Attacks Through the Lense of an NGO,” in 23rd USENIX Security Symposium (USENIX Security 14). San Diego, CA: USENIX Association, 2014, pp. 543–558.

[12] I. Santos et al., “Idea: Opcode-Sequence-Based Malware Detection,” in Engineering Secure Software and Systems, ser. Lecture Notes in Computer Science, F. Massacci, D. Wallach, and N. Zannone, Eds. Springer Berlin / Heidelberg, 2010, vol. 5965, pp. 35–43.

 ISBN: 978-1-61208-393-3 