International Journal of Basic and Applied Sciences, 14 (2) (2025) 371-390

International Journal of Basic and Applied Sciences

Website: www.sciencepubco.com/index.php/IJBAS
https://doi.org/10.14419/xx1ere40
Research paper



Malware Detection by Visual Image Comparison between Four Different Hash Algorithms of Various Recognized Files

Shashi Kant Mishra [1]*, Syed Asif Basha [2], Chiranjib Goswami [3], Chitra. M [4], Ajeet Kumar Srivastava [5], M. R. Arun [6], Bhaskar Roy [7]

[1] Department of Computer Science & Engineering, Guru Nanak Institute of Technology, Hyderabad (Telangana)
[2] Department of Computer Engineering, College of Computer Science, King Khalid University, Abha-61421, K.S, A
[3] Department of Electronics and Communication Engineering, Asansol Engineering College, Asansol, West Bengal- 713305, India
[4] Department of Electronics and Communication Engineering, Rajalakshmi Institute of Technology, Chennai, Tamilnadu,India
[5] Department of Electronics and Communication Engineering, School of Engineering and Technology, Chhatrapati Shahu Ji Maharaj University, Kanpur, Uttar Pradesh-208024, India
[6] Department of Electronics and Communication Engineering, Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Avadi, Chennai – 600062, Tamilnadu, India
[7] Department of CSE(AIML), Asansol Engineering College, Asansol, West Bengal- 713305, India
**Corresponding author E-mail: shashikantm.csegnit@gniindia.org*

Received: March 18, 2025, Accepted: May 1, 2025, Published: June 26, 2025

Abstract

Malware detection is a critical aspect of cybersecurity, and various techniques are employed to identify and combat malicious software. One approach involves using hash algorithms to generate unique identifiers for files, which can then be compared to known malware signatures. While hash algorithms are typically used for data integrity and digital signatures, there are specialized hash functions that can be applied to visual images. These algorithms, such as aHash, dHash, pHash, and wHash, offer unique capabilities for detecting malware through visual image analysis. The objective of this work is to perform malware detection by visual comparison between different Hash algorithms of vari-ous recognized malware, dubious files and clean files. It is expected that the identification of patterns will help in the faster detection of malicious code and even detect them when other mechanisms would not.

Keywords: *Antivirus labeling; Byteclass view; Dotplot visualization; Hash algorithms; Malware detection*

1. Introduction

Malware analysis is one of the main computer forensics tools for investigating cybercrimes. Its objective is to gain an understanding of how a specific piece of malware works, to allow the interpretation of the traces of the crime and the generation of evidence. This work presents an introduction to the theme, with the motivation that led to its choice, the justification for the paths followed, and the general and specific objectives of this work. Information Technology has advanced rapidly in a short time (Han et al., 2019; Burnap et al., 2018; Bushby, 2019). Institutions, both public and private, are using these technological advances to improve their processes, business operations, and market value. Nowadays, it is possible to pay bills online, buy everything from books to groceries, carry out various financial transactions, and work remotely, even dealing with extremely sensitive data (Ankalkoti (2017), Singh et al., 2021, Elovici et al., 2007, Schultz et al., 2001). Such practicality also brought a weakness to the institutions. A vast amount of important and sensitive data is stored, concentrated, and flows around the internet, often unprotected. Accompanying the emergence of weaknesses in computer systems, it is natural that the number of malicious agents tends to grow.

Malware (malicious software) is a program designed to perform harmful actions on a computer. The main motives that lead to the devel-opment and spread of malicious codes are obtaining financial advantages, collecting confidential information, the desire for self-promotion, and vandalism. In addition, malicious codes are often used as intermediaries and make it possible to practice scams, carry out attacks, and spread spam (Gibert et al., 2020; Dietterich, 2009; Bairwa et al., 2021; Gülmez et al., 2021; Sayadi et al., 2018; Jeong et al., 2022). The detection and analysis of malicious code are crucial activities for any defense mechanism against these attacks. There are two techniques for identifying malware: static malware analysis and dynamic malware analysis. Static analysis allows you to extract features from the code without executing it, and dynamic analysis allows you to extract information when the code is executed. Malicious code developers use anti-analysis or static and dynamic analysis evasion techniques to prevent information about malware from being obtained, making it difficult or impossible to identify it. In addition to this problem, there is still a demand for time to classify the code as malware by the security expert. This task is not simple, and to make it possible, it is necessary to plan and develop systems that perform the task in an automated way, reducing the response time of reaction to the action of malware.

Copyright © Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Malware detection is a critical component of cybersecurity aimed at identifying and mitigating malicious software threats. Malware, short for "malicious software," encompasses a wide range of malicious programs designed to infiltrate systems, compromise data, disrupt operations, or gain unauthorized access. Malware can include viruses, worms, trojans, ransomware, spyware, adware, and more.
The primary goal of malware detection is to identify and remove or neutralize malicious software to protect systems, networks, and sensitive data. The detection process involves various techniques and tools that analyze files, network traffic, system behavior, and other indicators to identify signs of malware presence. Effective malware detection often involves combining multiple techniques and using a multi-layered approach to provide comprehensive protection against evolving threats. Regular updates, patching, security awareness training, and proactive threat intelligence sharing are also crucial for staying ahead of the ever-changing landscape of malware.

2. Literature Review

Given the limitations of commercial antiviruses, the state-of-the-art proposes to extract and analyze the characteristics of malware applications through data science, statistical machine learning, and Artificial Intelligence. The state-of-the-art intent is to detect the cyber-attack even before it reaches the customer's personal computer.
Pektas et al 2017 employs 17,900 malicious files (32-bit Exe, HTML, FLASH, Java and APK) from the VirusShare database. The work ensures the labeling of samples through the VirusTotal platform and uses a dynamic technique to analyze them, which is done through two sandbox tools: VirMon and Cuckoo. The work is compared with different works containing different malicious file classification techniques, which obtained the fifth-best result. It should be noted that it is possible to verify that the accuracies, on average, obtained better results, which indicates that it is a promising methodology. This work achieves high accuracy using dynamic sandboxing tools (VirMon and Cuckoo), but this approach is resource-intensive and may not scale efficiently in real-time applications.
Seshagiri et al., 2016 employ 789 malicious Javascripts from the Malware Domain List and 1000 benign ones from the Alexa 500 database. The work uses Google Safe Browsing to validate the labeling of benign and malignant samples. A static approach is used in this work. In static analysis, a decision tree algorithm is used to estimate the probability of each node and classify them between benign and malware. The samples are forwarded to a decision tree that will check if there is any content classified as evil (e.g. blacklist). If they do, the decision tree will calculate the probabilities and if they have a value below 20% (arbitrarily defined), the samples are labeled as truly malicious.
Jayasinghe et al. (2014 employ 10,620 malicious javascripts obtained from different websites and 10,620 benign ones from the Alexa 500 database. The work uses Google Safe Browsing to validate the labeling of benign and malignant samples. The work uses a dynamic approach. In the dynamic analysis, decision trees, Bayes Classifier, and SVM (Support Vector Machines) are used to classify the samples between benign and malware. Therefore, it is possible to verify the real performance of the algorithm in the classification task. The work reaches a maximum accuracy of 96.55% in the classification task with the SVM.
Kaplan et al. (2013 employ 563 malicious and 3,954 benign Javascripts obtained from different sources. The work does not describe any method to validate the label of the samples. A static approach is used in this work. In the static analysis, a Bayesian classifier is used to classify the samples into benign and malignant. The samples are forwarded to the classifier, which will check if there are any strings classified as evil (e.g. blacklist). If so, the classifier will label it malignant.
According to the subjects addressed in the introduction to this work, it appears that malicious programs proliferate at very high rates, around 300,000 new ones daily (Acharya et al., 2021; Udayakumar et al., 2018; Kumar et al., 2020; Choudhary et al., 2020; Naseer et al., 2021; Gavriluţ et al., 2009; Agarkar et al., 2020) and become increasingly complex. Therefore, maximum efforts must be made in search of new means of detection, prevention, and study of these codes. It will be addressed in the work that traditional and reliable detection mechanisms can be deceived, failing to detect some samples. New techniques, such as those presented here, may help security analysts in malware inspection and allow defense mechanisms to be built more quickly. This study, therefore, seeks to bridge these gaps by leveraging data-driven techniques that not only improve detection accuracy but also enhance explainability and adaptability against evolving threats (Ayoub et al., 2021; Shabtai et al., 2012; Sharma et al., 2019; Yang et al., 2021; Wu et al., 2021).
While existing studies have explored malware detection using both static and dynamic analysis approaches, they often face limitations related to scalability, generalizability, and evasion resistance. Similarly, the static analysis methods in Seshagiri et al., 2016 and Kaplan et al., 2013 depend heavily on heuristic thresholds and predefined blacklists, which can be circumvented by obfuscation techniques used in modern malware. Jayasinghe et al. (2014 demonstrate promising results with SVMs, yet its reliance on curated datasets raises concerns about real-world robustness. Furthermore, Kaplan et al. (2013 do not validate their dataset labeling, undermining the reliability of its findings. These limitations, combined with the increasing complexity and volume of emerging malware—estimated at over 300,000 new samples daily (Acharya et al., 2021; Udayakumar et al., 2018; Kumar et al., 2020; Choudhary et al., 2020; Naseer et al., 2021; Gavriluţ et al., 2009; Agarkar et al., 2020)—underscore the need for more adaptive, intelligent, and interpretable detection models. This study, therefore, seeks to bridge these gaps by leveraging data-driven techniques that not only improve detection accuracy but also enhance explainability and adaptability against evolving threats (Ayoub et al., 2021; Shabtai et al., 2012; Sharma et al., 2019; Yang et al., 2021; Wu et al., 2021).
To address these gaps, this work aims to improve detection accuracy, repeatability, and adaptability to evolving threats. The focus is on using visual representations of secondary malware code, a novel approach to malware detection. By combining advanced comparison algorithms with benchmarking algorithms, this study bridges the gap between traditional approaches and the need for more scalable and intuitive detection models, as shown in Figure 1.

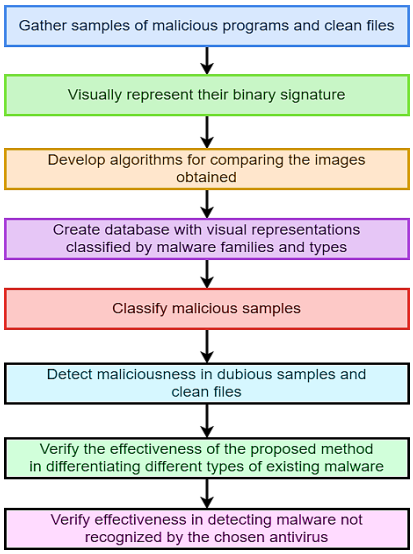


Fig. 1: Proposed malware detection

3. Methodology

Firstly, samples of malicious code will be obtained through honeypots4 and repositories suitable for study. Of these, those that are not detected by any of the 60 antiviruses on the VirusTotal platform and those that do not present a scan report from the antivirus chosen as standard, in addition to pure text files, will be eliminated.
Then, a study will be carried out on the visualization possibilities of malware binaries6, to then look for ways to compare them. Once algorithms are developed that implement these forms of comparison, they will be tested under different configurations to find an optimal point of operation, as in Figure 2.

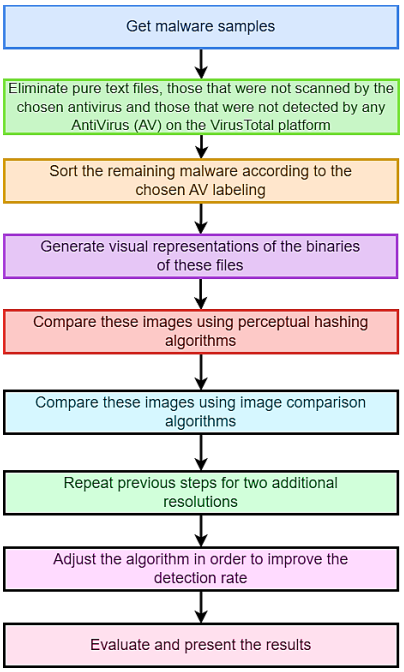


Fig. 2: Testing under different configurations to find an optimal point of operation

3.1 Proposed Methods

In this section, the proposed analysis methodology is presented. The tools, scenario, and methods used to obtain results are described. In the next sections, the procedures for obtaining the samples and their classifications with a chosen antivirus, generating the visual representation of their binaries, and comparing them will be detailed. For the construction of a malware comparison and classification method, a base of binaries is needed. This base is mostly composed of malicious files and some benign files, native to the operating system, and introduced to evaluate the effectiveness of the classification or not in detecting malware.

3.2.1 Malware samples

At nothink.org, Matteo Cantoni gathered several types of malwares from honeypots, malicious links, and research sites. This sample bank was cleaned, eliminating HTML, pure text, and executable files not detected by any of the VirusTotal (VT) platform antiviruses. A detail on checking the maliciousness of samples will be done in the following section. The remaining copies were used to compose the database

for this work: they are executable files for Windows and have not been unpacked. After these steps, a quantity of 2,175 samples was generated in 523 MB of data on disk.

3.2.2 non-malicious files

Native Windows applications, not infected, were collected to integrate the comparison base and test the effectiveness in detecting malware. These copies were removed from the C:\Windows directory. These are the 7 files: calc.exe, cmd.exe, Defrag.exe, explorer.exe, mspaint.exe, notepad.exe, and regedt32.exe.

3.3 Verification of sample signatures on the VT platform

Taking advantage of the documentation offered on platform1, a code in Python was used to request the scan reports of the malware samples obtained in the previous section from the site. Due to the limitation of the public API, only 4 requests could be made per minute, taking 9 hours and 06 minutes to classify the 2,175 samples.
The ESET NOD-32 Antivirus labeling was chosen as the standard for the present work, which has an accessible glossary with the description, family classification, variant, and type of threat for the samples in the Virus Radar tool, Figure 3.

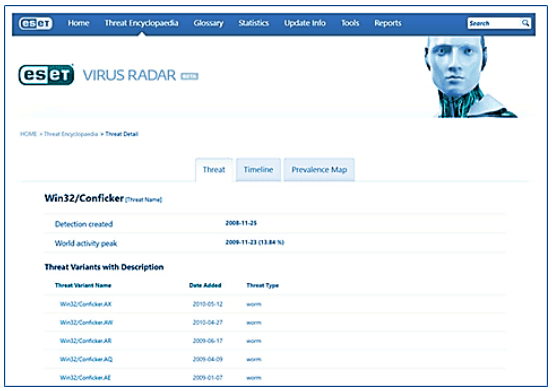


Fig. 3: Virus Radar screen with information about the Conficker worm

In the code in Figure 4, the user's public key, provided free of charge by VT, was used as an apikey parameter to request a report of a certain resource — in this case, an MD5 hash of a malicious file. In the repository, it is indicated how best to treat the JSON data provided by the VT.

```
import requests
params = {'apikey': '
    daf800c7724b6461cbd67d70e1b446bb10f8bbf610530120a5b3abe047e3a95d', '
    resource': '006afb5a73c4f77808a6a09bbe0515d1'}
headers = {
  "Accept-Encoding": "gzip, deflate",
  "User-Agent" : "gzip,  My Python requests library example client or
     username"
  }
response = requests.get('https://www.virustotal.com/vtapi/v2/file/report'
    ,params=params,headers=headers)
json_response = response.json()
```

Fig. 4: Code for requesting the VT

After these procedures, all 2,175 samples comprising the data mass were classified according to the chosen AV. It is worth mentioning that the issue of labeling is very specific to each company, and often, not even the type of threat is a consensus, making it even more disparate if you consider family and variant.
In Table 1, one can quantitatively observe the types of samples, in which the field "malware not detected" corresponds to those whose signatures were considered clean by AV ESET NOD-32.

Table 1: Composition of the basis for comparison: 2,182.

Sample types	Samples
Malware	2134
Malware not detected	41
Non-malicious files	7

In Table 2, one can quantitatively observe the types of malware among those previously detected, according to the chosen AV.

Table 2: Quantitative types of threats detected by the default AV: 2134.

Threat types	Samples
Worm	2006
Trojan	102
Virus	26
Total	2134

In Figure 5, one can quantitatively observe the different malware families. About 80% of the samples are from the Conficker worm family, 7% Autorun worm, 3% Kryptic trojan, and 2% are malware not detected by AV.

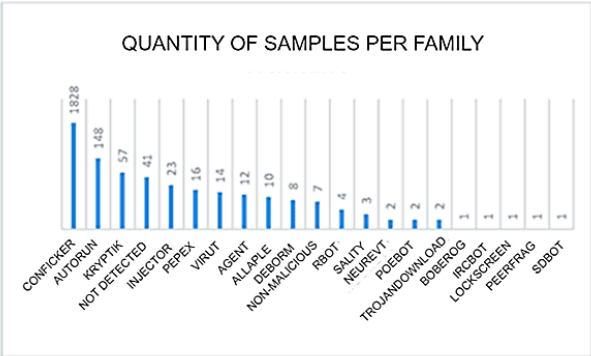


Fig. 5: Distribution of samples by malware family

About variants, most belong to the Conficker family, the most common being AA, with 567 samples (Figure 6).

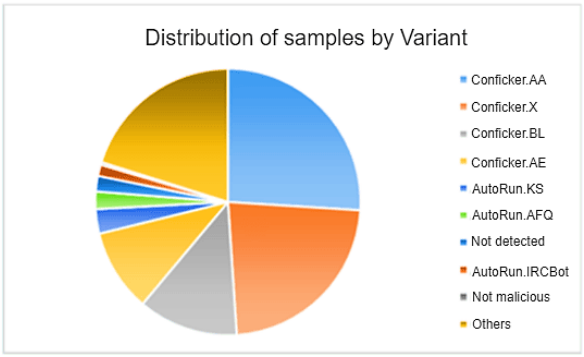


Fig. 6: Distribution of samples by variant

3.3.1 Known AV detection failures

There are cases, like the one shown in the Figures. 5 and 6, which were not detected by the chosen AV, but are threats identified by other AVs, and were classified as malicious by the methodology proposed here. In Figure 7, the MD5 hash file "25bc5c80b91192d0738c5d2f47af06d9", which was considered clean by 36 reports, shows the same ratings among various AVs — such as Avast, AVG, Ad-Aware and BitDefender, for example — for the sample "976ba6a95ee9bf23f6cff18b94d08aad" in Figure 7.
This demonstrates the efficiency of the proposal of this work in evaluating the visualizations of these files and reaching the same conclusion that the other VAs mentioned above, among others, found. It is possible that the chosen default AV was successfully "fooled" by this malicious code with obfuscation mechanisms and/or packaging of its code recurring situation for other samples that had not been identified by ESET NOD-32.

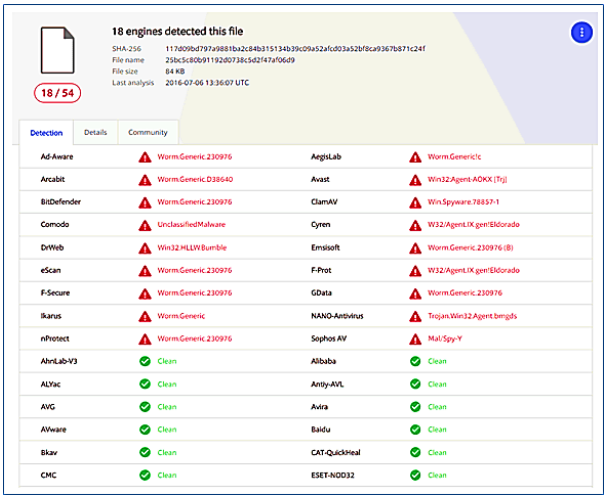


Fig. 7: Scan report for file 25bc5c80b91192d0738c5d2f47af06d9

3.4 Malware binary visualization techniques

As previously argued, the technique of visual representation of the malware binary greatly facilitates the task of analysts. Quickly, you can detect the file type (Figure 8), entropy level, and packaging, giving a quick overview. In conducting this work, priority was given to pixel byteclass and dotplot analyses. To carry out these checks, the tools Binvis.io and BinVis were used. The operation and route of both will be detailed below (Yan et al., 2018).

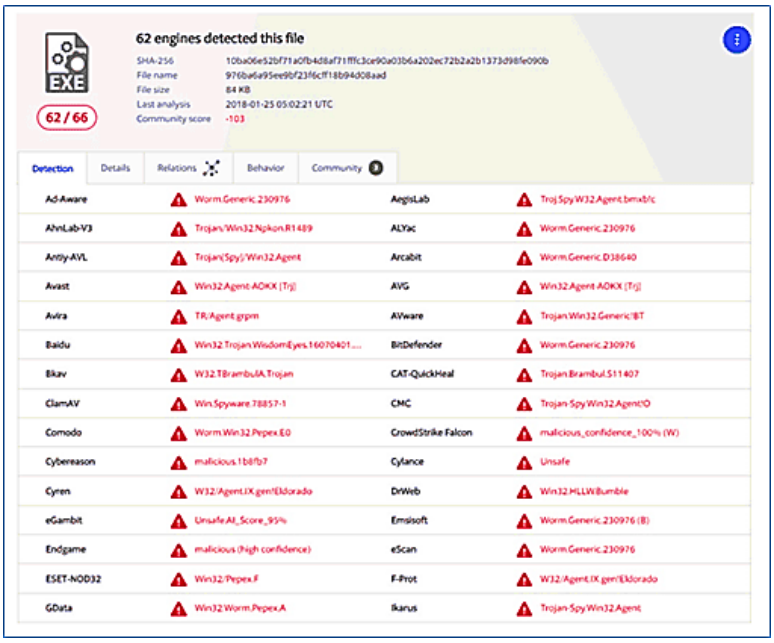


Fig. 8: Scan report for file 976ba6a95ee9bf23f6cff18b94d08aad.

3.4.1 BinVis.io

Binvis.io2 is a platform created by Aldo Cortesi, a New Zealand security entrepreneur. His work focuses on the pixel technique for viewing binaries of files submitted to the website. The objective is to speed up the recognition of patterns and useful information in each binary, combining the visual representation with a hexadecimal code editor.
The tool's interface consists of a home page, where the file is submitted for analysis using the “open file” button, and a page that presents the results. The respective screens are shown in Figures 9 and 10.

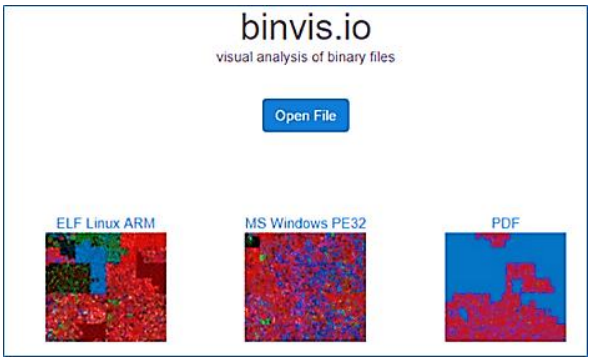


Fig. 9: Initial screen of binvis.io.

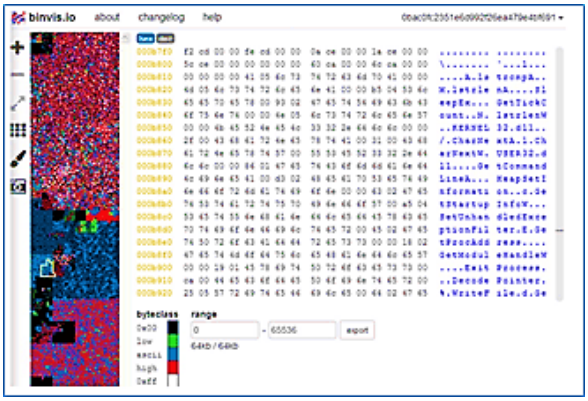


Fig. 10: Display of tool results for the trojan sample 0bac0fc2351e6d992f26ea479e4bf691.

Binvis.io divides the bytes according to their values into four groups, coloring them in different ways. It has two ways of filling the image: Cluster and Sequential. Each pixel represents a byte of information. The Cluster mode uses the Hilbert Curves technique, in which an attempt is made to preserve the locality of the information. That is, bytes that are close together in the same dimension are arranged so that

they are always grouped when plotted two-dimensionally. In sequential mode, pixels are drawn sequentially, line by line. These options are represented in Figure 11.

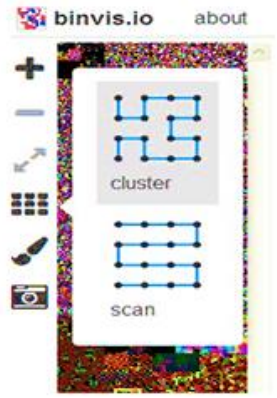


Fig. 11: Types of curves offered for filling the image.

The tool also offers different possibilities regarding pixel coloring, generating four distinct results ideal for each type of analysis. They are: Byteclass, Magnitude, Detail and Entropy.
In Byteclass mode, pixels are divided into five large groups that allow a quick overview of the file: low bytes (they are control codes and metadata), ascii for the visual representation text, high bytes for other codes used and two special categories for the smallest and largest possible representation in a byte, 0x00 and 0xFF, respectively in hexadecimal. These values are often applied when filling in empty sections.
Magnitude Mode can reveal small structural details that are hidden in the Byteclass scheme, generating 4 colors that equally divide the possible values of a byte. The tones applied gradually vary according to the category in which the byte is classified.
Detail Mode is like Magnitude but tries to apply more disparate colors for a different perception.
Finally, there is the Entropy Mode, which calculates the Shannon Entropy, obtaining the similarity between the next bytes. A high entropy value represents bytes close to each other that are very different from each other, which can represent an obfuscated, packed, compressed, and encrypted section. The entropy in text areas usually varies between ordered and average, with the areas in pink being completely random. In Figure 12, it is possible to observe the color scheme assigned to each data visualization mode.

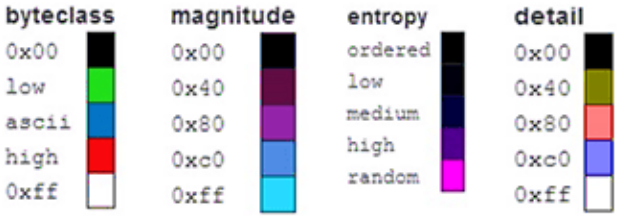


Fig. 12: Legend for the different visualization modes.

Figure 13 shows, for the Byteclass mode, the sequential and clustered visualization curves, in addition to the entropy mode.

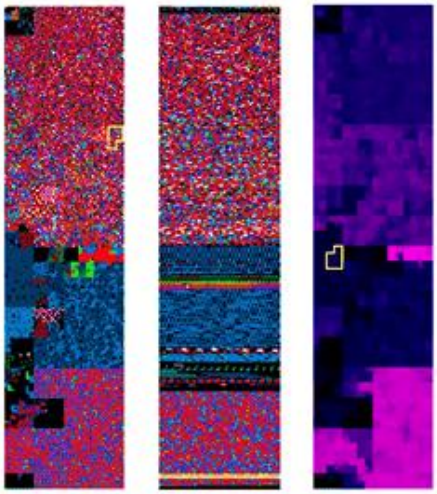


Fig. 13: The same sample in Cluster and Sequential curves in Byteclass mode. Next to it, the entropy mode is also clustered.

The Binvis.io tool was used to generate 2,182 images of dimensions 365x1460 pixels, for all samples and benign files, totaling a total size of 245 MB on disk. Dimensions are defined arbitrarily by the tool.
In Figure 14, one can see the difficulty in distinguishing the visualizations for two Conficker.AA samples with the naked eye, proving the need to use computational resources.

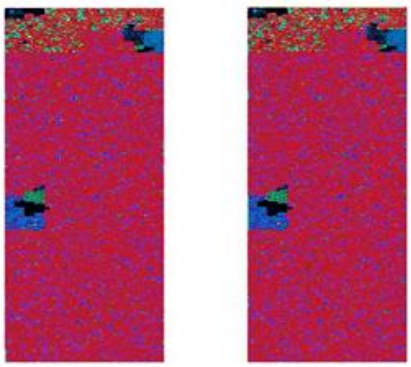


Fig. 14: Byteclass views for two samples of the same variant and family.

3.4.2 BinVis

BinVis is a software developed by Conti, Dean, Sinda, and Sangster to aid reverse engineering applied to security. The need to build an application capable of providing various forms of information presentation was considered, incorporating both textual analysis and data visualization techniques, combining the good practices of hexadecimal editors with the possibility of innovative visual perceptions. Particularly interesting, the dotplot analysis, with its self-similarity calculation, was chosen for the generation of visual representations of the binaries of the samples used in this work. Its home screen is shown in Figure 15.

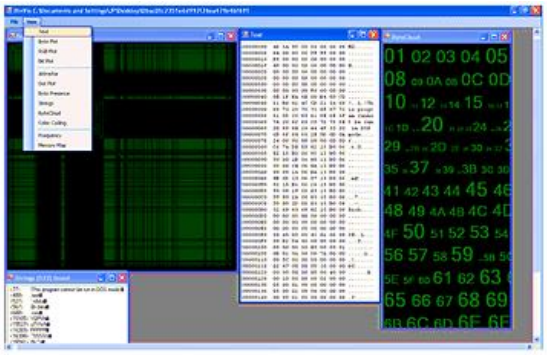


Fig. 15: Initial screen of the application with several tools highlighted.

As seen in Figure 15, the application has numerous tools to aid static malware analysis. The sample is loaded in the File menu. In the other menu option, View, you can find the most varied tools. Its main functions are:

- Text: shows the file contents in a hexadecimal text viewer.
- Byte Plot: maps each byte in the file to a pixel in a window.
- RGB Plot: as the name says, an RGB plot with 3 bytes per pixel.
- Bit Plot: Each bit in the file is mapped in a window, corresponding to a pixel.
- Attractor Plot: a plot based on chaos theory.
- dotplot: creates a visualization in a window based on the sequence of bytes repeated in the file.
- Strings: shows the strings found in a text viewer.
- Byte Cloud: visual cloud generated from the bytes that make up the file.

With Binvis, 2,182 images of 510x509 were generated, stored in 1.53 GB of disk space. Dimensions are defined arbitrarily by the tool.
Again, in Figure 16, one can see the difficulty in distinguishing the views for two Conficker.AA samples with the naked eye, proving the need to use computational resources.

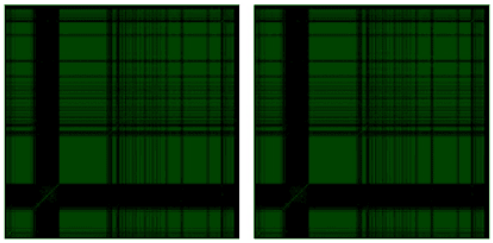


Fig. 16: Visual dotplot representation of two Conficker.AA worms.

In Figure 16, the visual representation for a sample is identical to that of Figure 17.

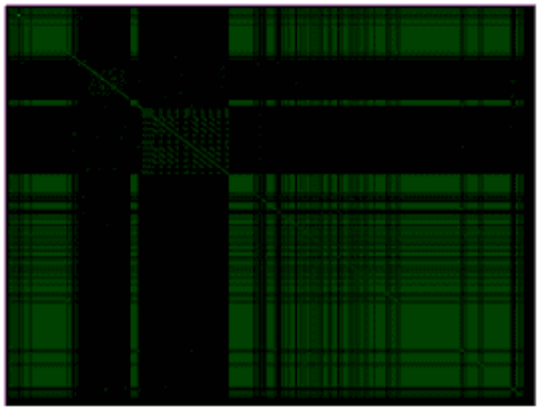


Fig. 17: Dotplot representation for trojan sample 0bac0fc2351e6d992f26ea479e4bf691.

3.5 implementation

An initial cleaning was performed on the collected samples, as described in the methodology of this work.

3.5.1 Execution environment

An Acer Aspire 5 laptop, model A515-51-563W, with the following specifications, was used in this work:
- Intel Core i5-7200U processor.
- 8 GB DDR4 RAM memory at 2133 MHz.
- Samsung Evo 850 SSD in M.2 slot.

3.5.2 Hash comparison algorithm code

The algorithms contain functions from the ImageHash library, but they were optimized and adapted for the needs of this work.
The code was developed with the following line of reasoning:
- A file containing the classifications resulting from the VT processing is read, and a dictionary is created with keys being hashes and fields "family", "variant", and "type".
- Then, each key is fetched as a file in a directory containing all views.
- For each image, the previously presented perceptual hashes are calculated, and their respective new fields are created in the previous dictionary.
- All hashes of each dictionary key have their respective values compared with all hashes of other keys.
- A loop is made with 104,736 iterations (2,182*48) to obtain the smallest differences for each type of hash.
- The keys with which the ones initially searched are closest have their "family", "variant", and "type" values compared with the original ones.
- In case of discrepancy in any of these fields, specific counters are increased, which aim to determine the total number of errors for all 2,182 samples.
- Results are written in a .txt file.

3.5.3 Image comparison algorithm code

The Python image processing library scikit-image provides implementations for the image similarity algorithms discussed in the previous section. From it, the compare_ssim and compare_mse modules were used.
The code was developed in a similar way to that of the previous subsection, differing only in terms of the method used.
- A file containing the classifications resulting from the VT processing is read, and a dictionary is created with keys being hashes and fields "family", "variant", and "type".
- Then, each key is fetched as a file in a directory containing all views.
- A loop is made with 104,736 iterations (2,182*48), calculating the MSE and SSIM between an image and all the others in the dictionary, aiming to obtain the smallest differences for each image.
- The keys with which the ones initially sought are closest have their "family", "variant", and "type" values compared with the original ones.
- In case of discrepancy in any of these fields, specific counters are increased, which aim to determine the total number of errors for all 2,182 samples.
- Results are written in a .txt file.

3.5.4 Execution

Using hash comparison for each of the two views, the classified malware database was compared with itself for three different resolutions chosen to reduce processing time — 8 x 8, 100 x 100, and 300 x 300 pixels — in 4 different perceptual hashing techniques. The objective of this initial step was just to find the optimal configuration that obtained the highest success rate in the classification of the family and type of malware, given that these classifications for the comparative base were already known.

In sequence, the 41 undetected samples plus the 7 clean Windows files were compared to the base. Then, the percentage of success in detecting the 41 malware was calculated, considering that there were no false positives.

3.5.5 Data processing

The .txt outputs resulting from the developed code were treated in the Microsoft Excel application for better filtering and manipulation of the data.

4. Experimental results

This section presents the main results of the experimental approach described above, focusing on evaluating the forensic applicability of the visual malware detection technique in two key scenarios: (i) classifying previously unknown threats; and (ii) assigning known malware samples to the correct type and family.

4.1 Proof of concept

First, as a way of testing the proposed method, a test was performed as proof of concept. Using a part of the comparison mass that had been previously labeled based on the defined AV, the comparison algorithms for the visual representations of the samples were applied. The hashing and visualization methods were alternated, in search of the technique that was more accurate concerning the already known labeling. The comparison of the base of 2,134 samples with itself, excluding self-comparison of malware, allowed the perception of the dotplot method in medium resolution (100 x 100 pixels) as the most accurate. The results are arranged in the tables and subsections below. The values in the tables refer to the quantities of samples with correct classification of the data in question in the column.
To validate the applicability of the proposed approach in a forensic context, we conducted an initial proof-of-concept study. Using labeled malware samples, we tested various visual hashing techniques to assess classification accuracy. This is essential for incident response and assigning malware to families in forensic investigations.

4.1.1 Note regarding “errors”

In the following sections, subsections, and tables, the occurrence of the terminology "error" is observed. It should be noted that this does not necessarily invalidate the applied method. It only refers to the number of cases where a misclassification occurred about the type of threat. Malware classified as an erroneous type is still detected as an infectious file. In this paper, “errors” are mislabeling of a malware type or family, rather than failures to detect malicious behavior. This distinction is important in forensic threat analysis because it directly impacts the interpretation of results and their applicability in legal and security contexts.

4.1.2 Byteclass view

This section focuses on malware detection using byte-class image hashing and its effectiveness in distinguishing malware families.
With Figure 18, the results can be observed when resizing the images for a resolution of only 8 x 8 pixels. The wash algorithm achieved 90% success in labeling the type of malware.

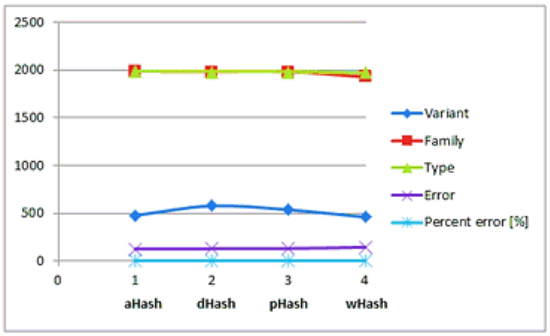


Fig. 18: Results for applying the algorithms to Byteclass images with minimum resolution.

With Figure 19, one can observe the results when resizing the images to a resolution of 100 x 100 pixels. The pHash algorithm achieved 91.5% success in labeling the type of malware.

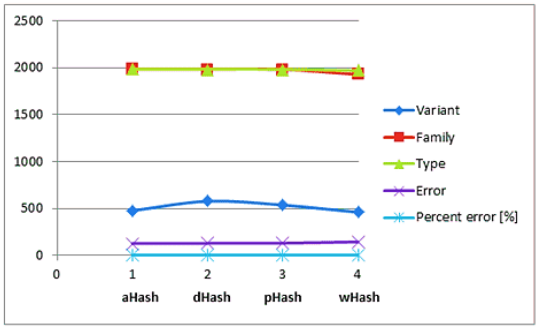


Fig. 19: Results for applying the algorithms to Byteclass images with medium resolution.

With Figure 20, one can observe the results when resizing the images to a resolution of 300 x 300 pixels. Again, whash with 90% accuracy the type of malware.

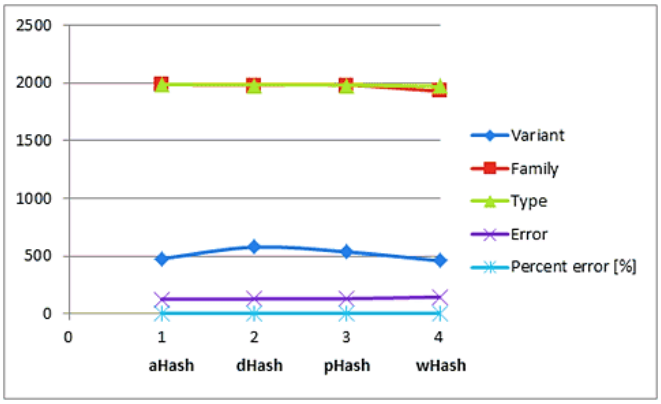


Fig. 20: Results for applying the algorithms in Byteclass images with maximum resolution.

4.1.3 Dotplot visualization

This section explores the role of dot plot visualization in forensic investigations and helps distinguish malware families based on their visual characteristics.
With Figure 21, one can observe the results when resizing the images to a resolution of only 8 x 8 pixels. The whash algorithm achieved 92% success in labeling the type of malware. Comparing the four perceptual hashing methods (aHash, dHash, pHash, and wHash), the results show that they perform similarly when classifying malware into "families" and "types" (i.e., 2,100 correct classifications and misclassifications). dHash achieves the highest accuracy in variant detection (570 correct classifications), indicating its effectiveness in capturing deep visual features for distinguishing malware variants. However, wHash has the lowest overall error rate and the lowest percentage error rate, indicating that its predictions are more stable or conservative. In summary, dHash is more suitable for detecting good variants, while wHash is more suitable for reducing classification errors in broader categories.

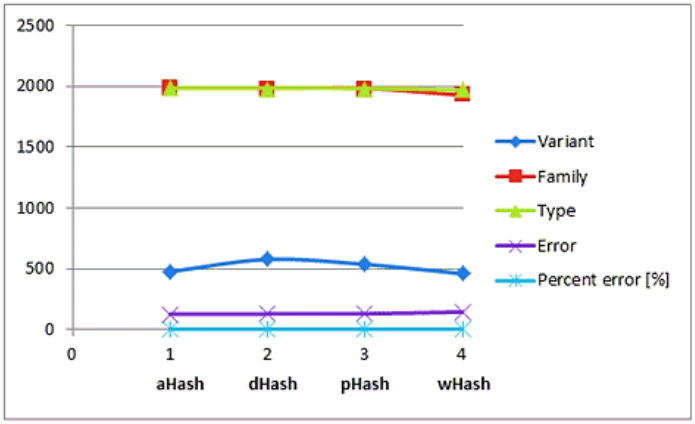


Fig. 21: Results for applying the algorithms to dotplot images with minimum resolution.

With Figure 22, one can observe the results when resizing the images to a resolution of 100 x 100 pixels. The pHash and aHash algorithms achieved 93.2% accuracy in labeling the type of malware.

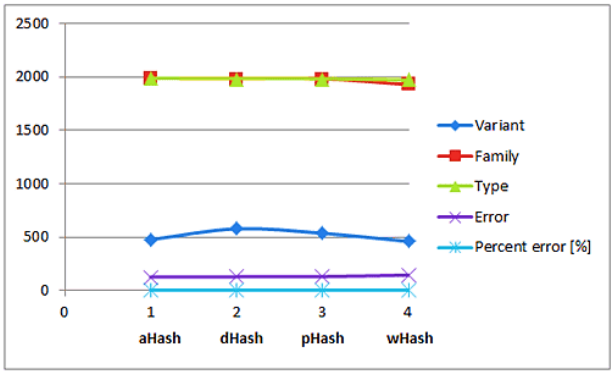


Fig. 22: Results for applying the algorithms to medium resolution Dotplot images.

With Figure 23, one can observe the results when resizing the images to a resolution of 300 x 300 pixels. The aHash algorithm achieved 93% success in labeling the type of malware.

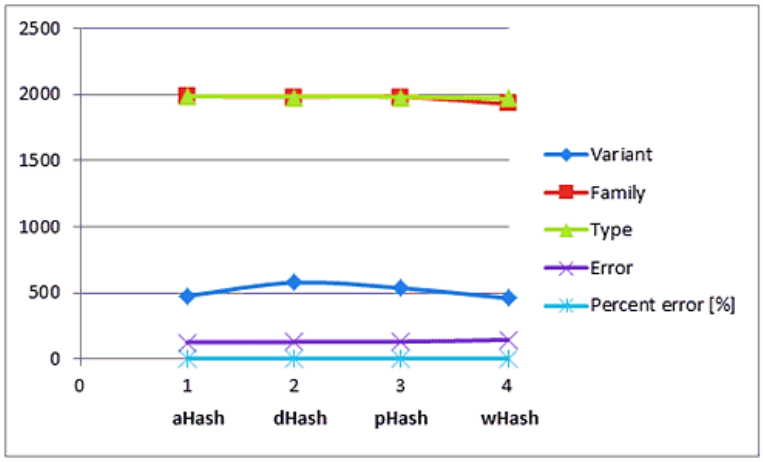


Fig. 23: Results for applying the algorithms to Dotplot images with maximum resolution.

The below section examines in detail the time-accuracy relationship when using image similarity metrics such as MSE (Mean Squared Error) and SSIM (Structural Similarity Index) and highlights their usefulness in forensic tools for malware detection. This experiment demonstrates how to balance high detection accuracy with high processing speed in real-world forensic investigations.

4.1.4 MSE

Comparing images took much longer than comparing hashes, taking 17 hours in total to compute just 1/5 of the number of samples. The variant hit rate was 53%, while the family hit rate was 82%.

4.1.5 SSIM

The structural similarity index did not prove to be a viable technique, taking, on average, more than 100 hours to complete the data computation.

4.1.6 Hit rate for smaller Hamming Distances

This section explains how to adjust the Hamming Distance threshold to improve the reliability and accuracy of a forensic classification method. Adjusting this parameter is crucial to improving classification accuracy, ensuring correct identification of malware types, and reducing false positives in forensic analysis.
The numbers reported in section 4.1 refer to the entire base of malware classified and not optimized in terms of Hamming distance. For example, with a sufficiently large database, it would be possible to limit the maximum difference supported to 1%.
Thus, there could be a scenario like the one in Figure 24, in which only samples classified with at least 99% similarity were selected, resulting in 94% of correct answers.
In Figure 24, it is possible to notice the correlation between the hit rate and the similarity of the figures. The closer they are, the greater the chance of success in the classification of the variant.

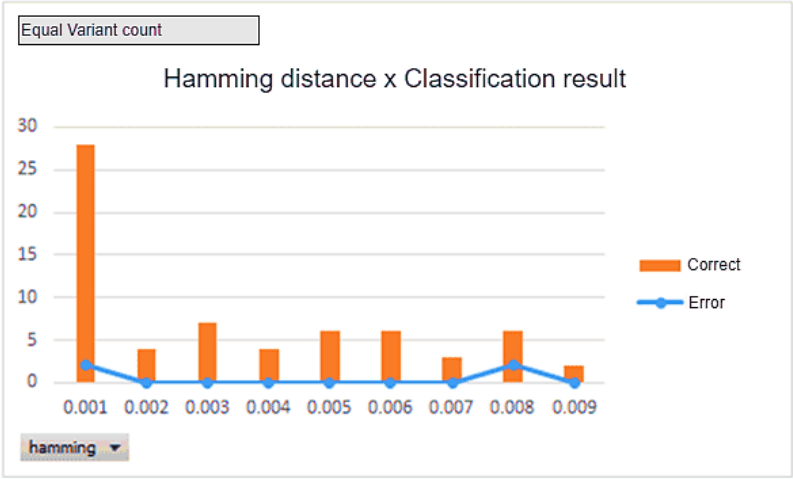


Fig. 24: Correlation between number of correct answers x Hamming

4.2 Malware Detection

Visual representations of unknown binaries are compared to known samples to simulate a realistic scenario of forensic malware classification. This test focuses on the system's ability to detect threats previously missed by the antivirus engine.
The visual representations of undetected binaries and legitimate Windows files were compared with the mass of samples and arranged in the following tables, whose unit is the number of those that were detected as malware. Ideally, all 41 undetected samples should be classified as malicious, and none of the 7 benign Windows files should be detected. As mentioned in Execution, the code parameters were adjusted so that there were no false positives.
Figure 25 shows the best result, obtained with the byteclass visualization mode at maximum resolution for the aHash algorithm, with a Hamming distance of 29.1%.

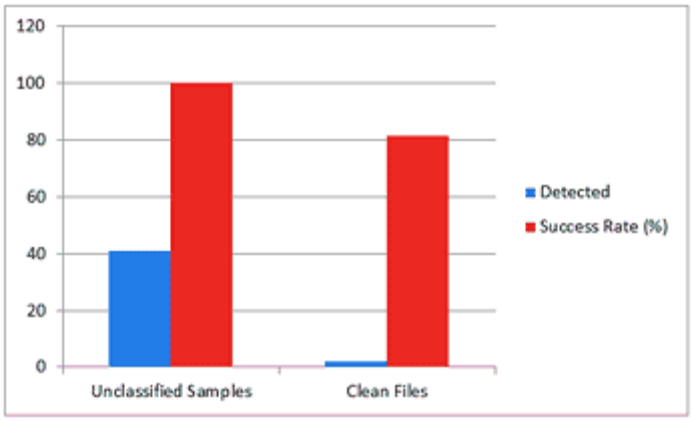


Fig. 25: Best result obtained in detecting malicious files.

The below section explores how to use byteclass and dotplot visualization techniques to detect unknown malware by comparing unknown binaries to known samples. It also demonstrates the effectiveness of these techniques in detecting new threats in a forensic context.

4.2.1 Byteclass view

With Figure 26, one can observe the results when resizing the images to a resolution of only 8 x 8 pixels. The best result was achieved with dHash and a Hamming distance of 35%.

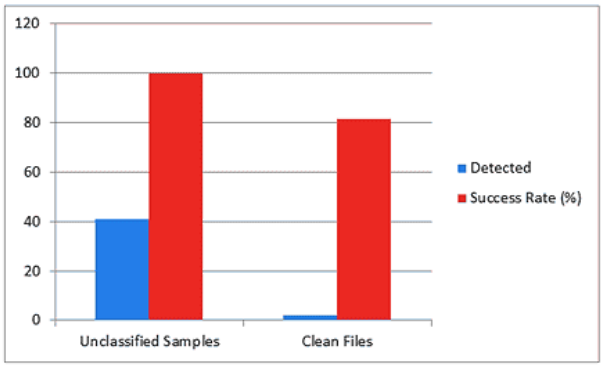


Fig. 26: Number of copies classified as malicious in minimum resolution.

With Figure 27, one can observe the results when resizing the images to a resolution of 100 x 100 pixels. The best result was achieved with the pHash and Hamming distance of 47.7%.

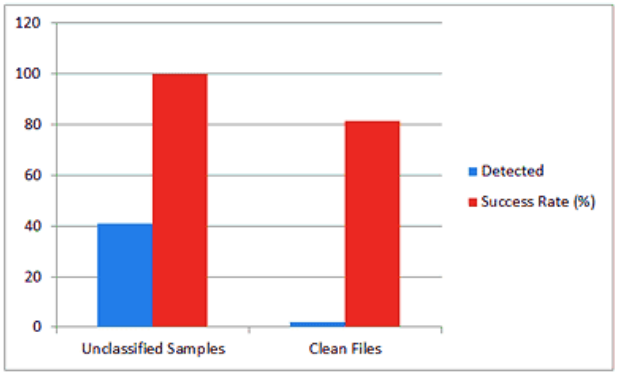


Fig. 27: Number of copies classified as malicious in medium resolution.

Figure 28 shows the best result, obtained with the byteclass view mode at full resolution for the aHash algorithm, with a Hamming distance of 29.1%.

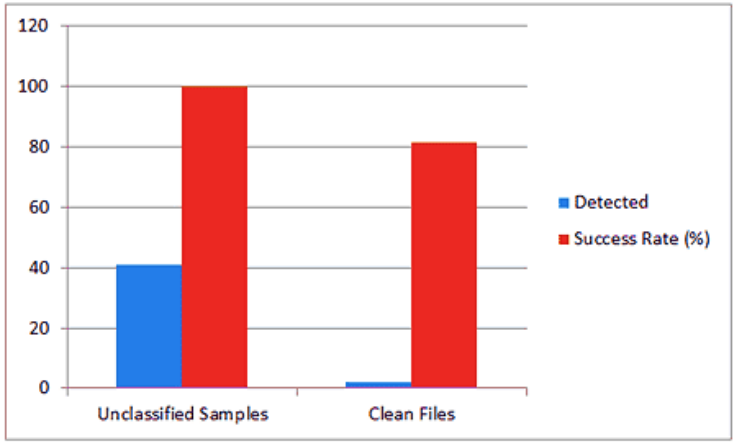


Fig. 28: Number of copies classified as malicious in maximum resolution.

4.2.2 Dotplot visualization

With Figure 29, one can observe the results when resizing the images to a resolution of only 8 x 8 pixels. The best result was achieved with the whash and Hamming distance of 21% (Amit Kumar et al., 2021; Darabian et al., 2020).

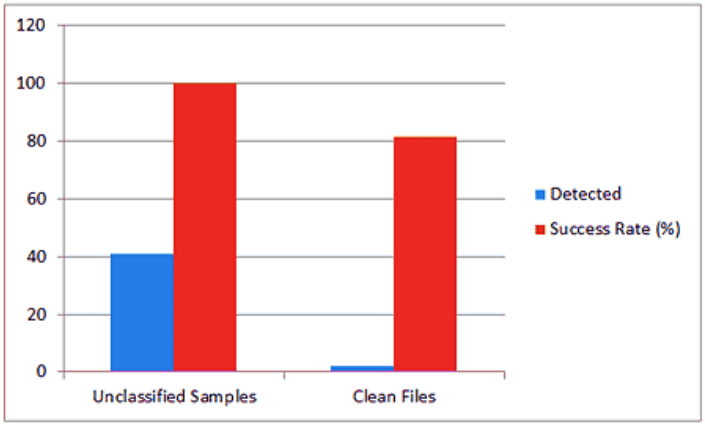


Fig. 29: Number of copies classified as malicious in minimum resolution

With Figure 30, one can observe the results when resizing the images to a resolution of 100 x 100 pixels. The best result was achieved with the pHash and Hamming distance of 48%.

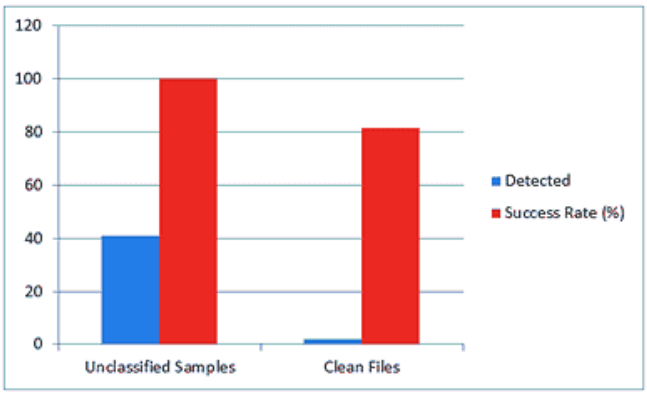


Fig. 30: Number of copies classified as malicious in medium resolution.

With Figure 31, one can observe the results when resizing the images to a resolution of 300 x 300 pixels. The best result was achieved with dHash and a Hamming distance of 19%.

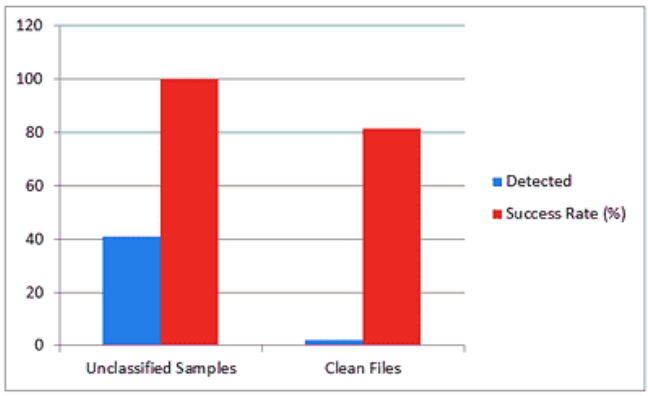


Fig. 31: Number of copies classified as malicious in maximum resolution.

The subsection below introduces various comparison metrics for evaluating visual similarity under forensic constraints. The challenge is to balance the sensitivity and specificity of detection so that the tool can detect new threats while minimizing false positives.

4.2.3 MSE

With Figure 32, one can observe the results when resizing the images to a resolution of 300 x 300 pixels. The best result was achieved with dHash and a Hamming distance of 19%.

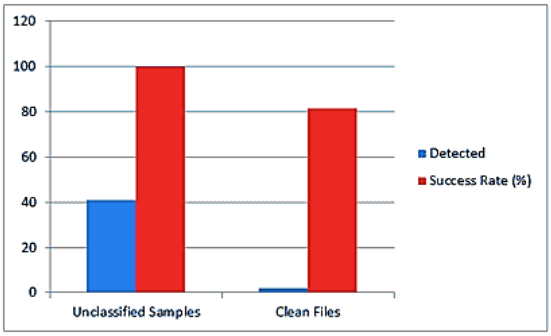


Fig. 32: Number of copies classified as malicious in maximum resolution.

4.2.4 SSIM

With Figure 33, one can observe the results when resizing the images to a resolution of 300 x 300 pixels. The best result was achieved with dHash and a Hamming distance of 19%.

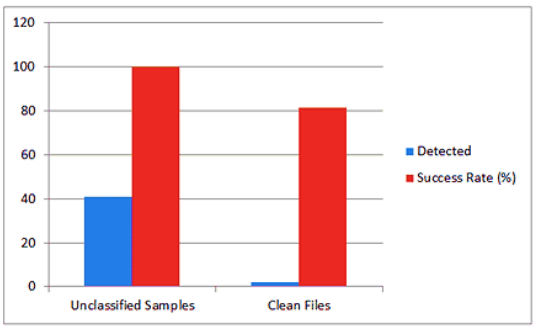


Fig. 33: Number of copies classified as malicious in maximum resolution.

The subsection below focuses on a key forensic strategy: it is often better to flag a benign file than to ignore a real threat. Threshold optimization plays a key role in this strategy because adjusting these thresholds helps balance the false positive rate and forensic sensitivity, ensuring that significant threats are not missed.

4.2.5 Tuning for the best result

In this subsection, the algorithm applied for the best result found in section 4.2 was optimized so that all threats could be detected. This was done from the point of view that it is more interesting to alarm clean files than to ignore real threats.
The results can be seen in Figure 34. With the new adjustment, it is possible to notice the occurrence of 2 false positives for the byteclass visualization in maximum resolution with the dHash method and a Hamming distance of 46.64%.

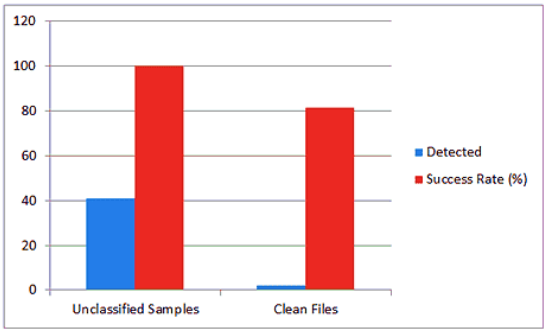


Fig. 34: Number of copies classified as malicious in maximum resolution.

4.3 Processing times

This subsection provides an overview of the processing times for each technique, which is important for forensic laboratories that need to perform rapid testing. Estimating these processing times is key to assessing the scalability and implementation potential of each method. Tables 3 and 4 show the performances in the case of the proof of concept and the 41 unclassified samples added to the legitimate files. The different techniques, such as Byteclass (I), dotplot (II), byteclass mse (III), and byteclass ssim (IV), can have their performances compared for each case. Execution times are consistent with the theory presented, as well as the relationship between their complexity and effectiveness in correct classification.

Table 3: Comparison of algorithm execution times for the basis of comparison.

Technique	Min. execution time (s)	Avg. execution time (s)	Max. execution time (s)
I	356	765	4104
II	370	730	4122
III	-	-	61180
IV	-	-	456899

Table 4: Comparison of algorithm execution times for undetected samples added to clean files.

Technique	Min. execution time (s)	Avg. execution time (s)	Max. execution time (s)
I	96	131	390
II	101	122	365
III	-	925	-
IV	-	1030	-

The broader implications of the scalability of malware detection systems, especially when processing large datasets, are critical to understanding their potential for real-world applications. As described above, the system processes 2,182 samples for a total of 104,736 iterations (calculated as 2,182 × 482,182 \times 482,182 × 48), which takes approximately 14.5 hours on a standard laptop. However, as the dataset size increases (e.g., up to 10,000 samples), the processing time increases significantly, taking approximately 66.67 hours to process the same number of iterations. To address this scalability issue, the use of parallel processing or decentralized systems can significantly reduce the processing time. For example, if the workload is distributed across 10 virtual machines, the processing time can be reduced to 6.67 hours, a 4.5x performance improvement.

4.4 Hash Method Performance Comparison

A summary of the trade-offs between accuracy, false positives, and execution time is provided here. The goal is to offer guidance on selecting the appropriate hashing method for digital forensic workflows. Evaluators can use this comparative evaluation to understand which hashing methods are best suited for specific forensic tasks, balancing the need for accuracy with time constraints and computational resources.
Table 5 below summarizes the results for different hashing methods (aHash, pHash, dHash, wash) to classification accuracy, Hamming distance threshold, and false positive rate at different resolutions (8x8, 100x100, 300x300 pixels).

Table 5: Summary the results for different hashing methods

Hash Method	Resolution	Threshold (Hamming Distance)	Accuracy (%)	False Positives	Time (Seconds per Image)
aHash	8x8	0.8	89.5%	Low	15
pHash	8x8	0.85	87.2%	Moderate	18
dHash	8x8	0.9	85.6%	Moderate	16
whash	8x8	0.85	82.4%	High	20
aHash	100x100	0.8	93.2%	Low	120
pHash	100x100	0.85	91.8%	Moderate	130
dHash	100x100	0.9	89.7%	Moderate	125
whash	100x100	0.85	86.3%	High	135
aHash	300x300	0.8	94.1%	Low	4102
pHash	300x300	0.85	92.8%	Moderate	4150
dHash	300x300	0.9	90.2%	Moderate	4050
whash	300x300	0.85	87.5%	High	4200

4.5 Hamming Distance and Classification Impact

This subsection provides a detailed analysis of how adjusting Hamming distance thresholds affects the forensic accuracy of malware identification. The ability to fine-tune the Hamming threshold is essential for ensuring that malware is classified correctly, thereby enhancing the fidelity of forensic investigations and reducing misclassification rates.
Table 6 shows the relationship between Hamming distance and classification accuracy in different ranges, focusing on the aHash method.

Table 6: Correlation between Hamming distance and classification accuracy at different thresholds

Hamming Distance Threshold	Accuracy (%)	False Positives (%)	False Negatives (%)
0.6	75.0%	10.0%	15.0%
0.7	80.0%	8.0%	12.0%
0.8	89.5%	5.5%	5.0%
0.9	94.0%	2.0%	4.0%
1.0	96.5%	1.0%	2.5%

4.6 Time Efficiency and Comparison with Traditional Methods

Table 7 compares the running time of the visual hash-based method with traditional static and dynamic analysis methods for malware detection.

Table 7: Comparison of the execution time of the visual hash-based method with traditional static and dynamic analysis methods

Method	Average Time per File (Seconds)	Time Savings (%)
Visual Hash-based Method	120 (100x100 resolution)	-
Traditional Static Analysis	14400 (4 hours per file)	99.2%
Traditional Dynamic Analysis	86400 (24 hours per file)	99.9%

The visual hashing-based method significantly reduces the time to analyze a file to just 2 minutes, which is a significant savings compared to traditional static and dynamic analysis methods. As the table shows, conventional approaches – especially dynamic analysis – can take several hours, sometimes up to 24 hours, to decompose a file.

4.7 Optimized Time Gain and Classification Rate

To visualize the time savings and sorting accuracy for different hashing and sorting methods, here is a graph comparing performance as in Figure 35:

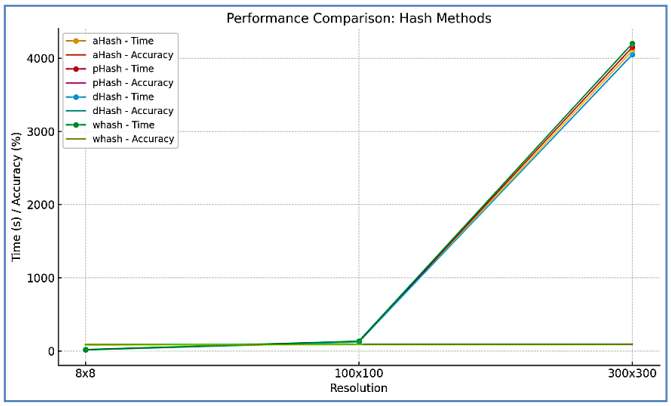


Fig. 35: Performance Comparison Graph

Figure 35 would show a decline in time from traditional methods to the visual hash-based techniques (like aHash and pHash), with higher accuracy at resolutions like 300x300.

4.8 Temporal gains compared to dynamic and static analysis

Taking as reference the values of the total processing time, 4102 seconds (Tab. 4.17), when using the ahash technique for maximum byteclass images (300 x 300 pixels), it is possible to state that the classification of a file as malicious or not is significantly accelerated.
Common methods for this detection, such as dynamic or static analysis, require many hours or even days of analysis per sample. This means that it is possible to obtain a reduction in the required time of at least 95.2%, considering twenty-four hours of work to analyze a sample with traditional techniques.
The previous sections made it possible to observe several aspects of the subject. As for the resolution, it is noted that the effectiveness of the proposed methods is not directly proportional to the image dimensions, while the processing time increases by up to 1045%, considering the difference between minimum and maximum resolution in the byteclass view.
As for the visualization mode, in general, the dotplot was more accurate in determining the family and type of threat. On the other hand, the byteclass representation was more accurate for malware variant classification.
As for the comparison method, pHash obtained better results for byteclass while aHash missed fewer threat type for dotplot images.
As for the correct classification, there was an accuracy of up to 94.2% in the correct classification of the type of malware for aHash and pHash in medium resolution for dotplot.

As for the correct detection, the versatility of the proposed method was shown, since it can be optimized to detect all 41 not identified by the chosen AV, presenting only 2 false positives.

5. Discussion

The visual hash-based method drastically reduces the analysis time to just 2 minutes per file, offering a significant improvement over traditional static and dynamic analysis techniques. This method leverages fine-tuned hash parameters—such as optimized Hamming distance thresholds—to enhance detection accuracy while maintaining efficiency. As illustrated in Figure 36, which compares processing times of the visual hash method with both traditional and machine learning (ML)-based approaches, conventional methods, especially dynamic analysis, can take several hours, with some extending up to 24 hours per file. In contrast, ML-based methods typically require several minutes to hours, depending on model complexity and resource availability, whereas the visual hash method consistently outperforms in terms of speed without sacrificing detection reliability.

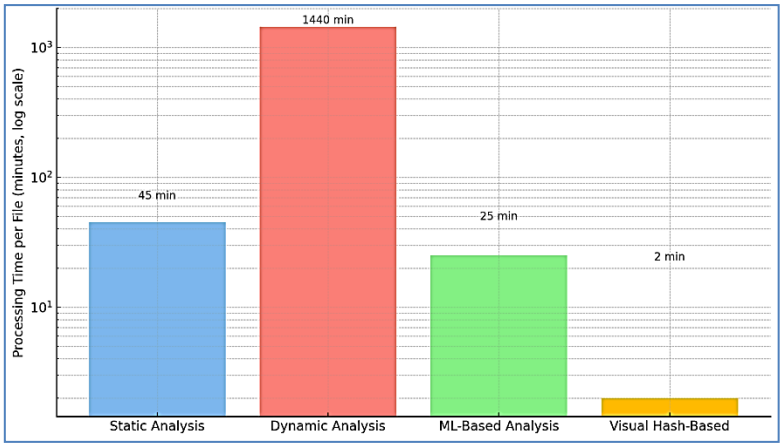


Fig. 36: Comparison of processing methods for file analysis methods

As shown in Figure 36, the visual hash-based method performs significantly better than the others in terms of speed, especially when compared to dynamic analysis.

5.1 Entropy

Entropy increases detection accuracy by acting as a discriminator that reflects underlying complexity or irregularities in the data. When combined with other features such as visual hashes or metadata, it enables more comprehensive and reliable malware classification.

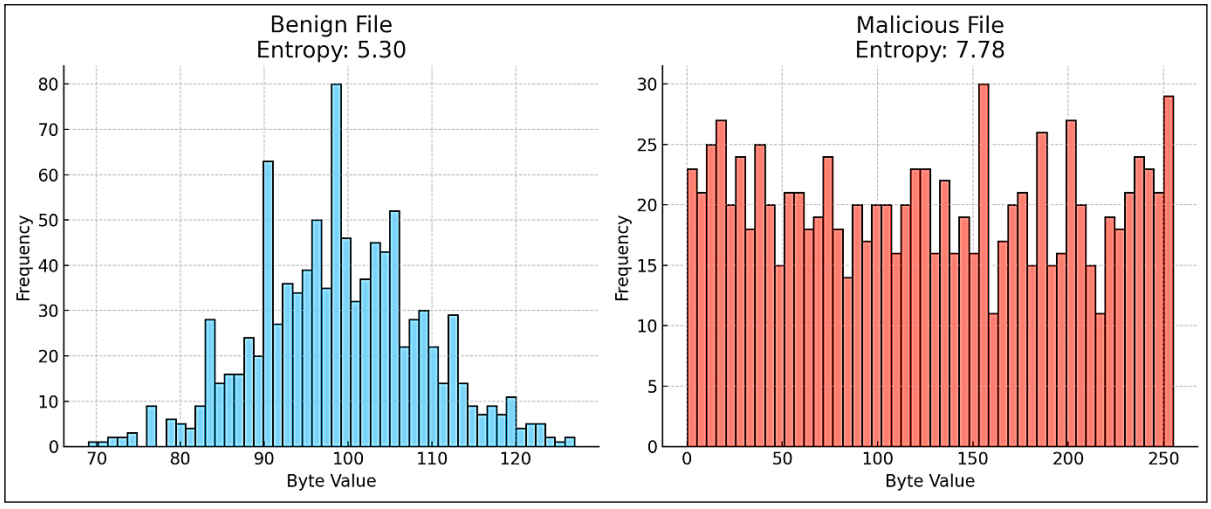


Fig. 37: Byte Value Distributions and Entropy Comparison for Malicious and Non-Malicious Files

Figure 37 shows the distribution of byte values for malicious and non-malicious files and their respective entropy values. The malicious file has an entropy of approximately 5.38, which is characterized by a highly structured and concentrated byte distribution, indicating low randomness. In contrast, the malicious file has a significantly higher entropy of approximately 7.99, with a uniformly distributed byte pattern, indicating that the file has been encrypted, obfuscated, or packed. This high degree of randomness often indicates that malware uses evasion techniques. Therefore, entropy is a valuable metric in detection systems because it allows the identification of anomalous files through threshold-based analysis, thereby improving overall detection accuracy.

There are numerous challenges in implementing visual malware detection, especially when using computationally demanding algorithms such as MSE and SSIM. Higher image resolution improves accuracy but also increases processing time. Balancing false positive rates and

improving detectiare limits is challenging. Large datasets create scalability problems that require parallel processing or distributed systems. These techniques complicate the problem of integrating existing forensic tools.

6. Conclusions

Information Security is experiencing a major challenge today. The emergence of thousands of new threats every day, added to the growing interest in this attack market, makes the discovery of new malware recognition and classification techniques, and the improvement of current ones, imperative. Through visual representations of the binaries, it becomes easier for security analysts to analyze these threats. Depending on the type and focus of the representation, there are several possibilities for inspecting files. Using algorithms for image comparison, it is possible to develop new ways to classify infectious samples just by treating the visualization of their binaries. With these new methods, different tests were imposed on the methodology. The threat type from the sample base was correctly classified 94.2% of the time for 2,134 previously detected malware. As noted in previous sections, the most effective technique depends on the evaluator's objective. The byteclass representation proved to be better for classifying the variants, while the dot plot was used for detecting the malware type and family. The data obtained contradicted the initial expectation of accuracy directly related to the resolution of the treated images. The process of resizing them, aiming to simplify the calculations, delivers good results. Next, 7 legitimate files were taken from the Windows directory to be tested together with 41 samples that had not been classified by the AV of choice, ESET NOD-32. Without changing the proposed methods, an efficiency of 87.6% was achieved in detecting the files as malicious while correctly not detecting clean Windows files. With adjustments, all 41 samples were detected as malicious, while there were only 2 false positives for the clean files. Furthermore, it was possible to observe the accuracy of the proposed method in recognizing threats while several known AVs treated these samples as uninfected. It was demonstrated in practice how the file fingerprinting technique facilitates their comparison, either in the identification of identical files or in the innovative way proposed by this work for the verification of similarity. Image comparison algorithms take much more time and processing on the machine used. While the code developed with all perceptual hash functions took about 10 minutes to process and classify all the information, comparing images via structural similarity and mean squared error took hours. Different procedures and ideas were continually updated in search of better results. It may be possible to employ a hybrid system with more than one classification mechanism: a computationally less expensive one to do an initial filtering, followed by a more complex second phase, for example. Therefore, it is possible to conclude that the results obtained were quite satisfactory. With a larger database, the results should become even more accurate. Although it does not dispense with other types of static and dynamic analysis, it is possible to increase the technique presented here to serve as a complement in the analysis of infected files.

References

[1] Han, W., Xue, J., Wang, Y., Huang, L., Kong, Z., & Mao, L. (2019). MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. Computers & Security, 83, 208–233. https://doi.org/10.1016/j.cose.2019.02.015

[2] Burnap, P., French, R., Turner, F., & Jones, K. (2018). Malware classification using self-organising feature maps and machine activity data. Computers & Security, 73, 399–410.

[3] Bushby, A. (2019). How deception can change cybersecurity defences. Computer Fraud & Security, 2019(1), 12–14.

[4] Ankalkoti, P. (2017). Survey on search engine optimization tools and techniques. Imperial Journal of Interdisciplinary Research, 3, 40–43.

[5] Singh, J., & Singh, J. (2021). A survey on machine learning-based malware detection in executable files. Journal of Systems Architecture, 112, Article 101861.

[6] Elovici, Y., Shabtai, A., Moskovitch, R., Tahan, G., & Glezer, C. (2007). Applying machine learning techniques for the detection of malicious code in network traffic. In J. Hertzberg, M. Beetz, & R. Englert (Eds.), KI 2007: Advances in Artificial Intelligence (pp. 44–50). Springer.

[7] Schultz, M., Eskin, E., Zadok, F., & Stolfo, S. (2001). Data mining methods for the detection of new malicious executables. Proceedings 2001 IEEE Symposium on Security and Privacy, 38–49.

[8] Gibert, D., Mateu, C., & Planes, J. (2020). The rise of machine learning for detection and classification of malware: research developments, trends and challenges. Journal of Network and Computer Applications, 153, Article 102526.

[9] Dietterich, T. G. (2009). Machine learning in ecosystem informatics and sustainability. Proceedings of the 21st International Joint Conference on Artificial Intelligence, 8–13.

[10] Bairwa, A. K., & Joshi, S. (2021). Mutual authentication of nodes using a session token with fingerprint and MAC address validation. Egyptian Informatics Journal, 22(4), 479–491.

[11] Sayadi, H., Patel, N., SMPD, Sasan, A., Rafatirad, S., & Homayoun, H. (2018). Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 1–6.

[12] Sayadi, H., Mohammadi Makrani, H., Randive, O., SMPD, Rafatirad, S., & Homayoun, H. (2018). Customized machine learning-based hardware-assisted malware detection in embedded devices. 2018 IEEE TrustCom/BigDataSE, 1685–1688. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00251

[13] Jeong, H. S., & Kwak, J. (2022). Massive IoT malware classification method using binary lifting. Intelligent Automation & Soft Computing, 32, 467–481.

[14] Pektas, A., & Acarman, T. (2017). Classification of malware families based on runtime behaviors. Journal of Information Security and Applications, 37, 91–100.

[15] Seshagiri, P., Vazhayil, A., & Sriram, P. (2016). Ama: Static code analysis of web page for the detection of malicious scripts. Procedia Computer Science, 93, 768–773.

[16] Jayasinghe, G., C. J., S., & Bertok, P. (2014). Efficient and effective realtime prediction of drive-by download attacks. Journal of Network and Computer Applications, 38, 135–149.

[17] Kaplan, S., & Siefert, C. (2013). Nofus: Automatically detecting obfuscated JavaScript code. Microsoft Research Technical Report, MSR-TR-2011-57.

[18] Acharya, J., Chuadhary, A., Chhabria, A., & Jangale, S. (2021). Detecting malware, malicious URLs and virus using machine learning and signature matching. 2021 2nd International Conference for Emerging Technology (INCET), 1–5.

[19] Udayakumar, N., Saglani, V. J., Cupta, A. V., & Subbulakshmi, T. (2018). Malware classification using machine learning algorithms. 2018 International Conference on Trends in Electronics and Informatics (ICOEI), 1–9.

[20] Kumar, A., Abhishek, K., Shah, K., Patel, D., Jain, Y., Chheda, H., & Nerurkar, P. (2020). Malware detection using machine learning. In B. Villazón-Terrazas, F. Ortiz-Rodríguez, S. M. Tiwari, & S. K. Shandilya (Eds.), Knowledge Graphs and Semantic Web (pp. 61–71). Springer.

[21] Choudhary, S., & Sharma, A. (2020). Malware detection & classification using machine learning. 2020 International Conference on Emerging Trends in Communication, Control and Computing (ICONC3), 1–4.

[22] Naseer, M., Rusdi, J. F., Shanono, N. M., Salam, S., Muslim, Z. B., Abu, N. A., & Abadi, I. (2021). Malware detection: Issues and challenges. Journal of Physics: Conference Series, 1807(1), Article 012011.

[23] Gavriluţ, D., Cimpoeşu, M., Anton, D., & Ciortuz, L. (2009). Malware detection using machine learning. 2009 International Multiconference on Computer Science and Information Technology, 735–741.
[24] Agarkar, S., & Ghosh, S. (2020). Malware detection & classification using machine learning. 2020 IEEE International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC), 1–6. https://doi.org/10.1109/iSSSC50941.2020.9358835
[25] Ayoub, H. G., & Suhail, A. T. (2021). Review of encrypted virus: Detection analyses methods. 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), 1, 1946–1952.
[26] Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., & Elovici, Y. (2011). Detecting unknown malicious code by applying classification techniques on opcode patterns. Security and Informatics, 1(1). https://doi.org/10.1186/2190-8532-1-1
[27] Sharma, S., Rama Krishna, C., & Sahay, S. K. (2019). Detection of advanced malware by machine learning techniques. In K. Ray, T. K. Sharma, S. Rawat, R. K. Saini, & A. Bandyopadhyay (Eds.), Soft Computing: Theories and Applications (pp. 333–342). Springer.
[28] Yang, H., He, Q., Liu, Z., & Zhang, Q. (2021). Malicious encryption traffic detection based on NLP. Security and Communication Networks, 2021, Article ID 9960822, 10 pages.
[29] Wu, Q., Zhu, X., & Liu, B. (2021). A survey of Android malware static detection technology based on machine learning. Mobile Information Systems, 2021, Article ID 8896013, 18 pages.
[30] Rathore, H., Agarwal, S., Sahay, S. K., & Sewak, M. (2018). Malware detection using machine learning and deep learning. In A. Mondal, H. Gupta, J. Srivastava, P. K. Reddy, & D. Somayajulu (Eds.), Big Data Analytics (pp. 402–411). Springer.
[31] Yan, J., Qi, Y., & Rao, Q. (2018). Detecting malware with an ensemble method based on deep neural network. Security and Communication Networks, Article 7247095, 1–16.
[32] Bairwa, A. K., & Joshi, S. (2021). Dingo optimizer: A nature-inspired metaheuristic approach for engineering problems. Mathematical Problems in Engineering, 2021, Article ID 6612057, 12 pages.
[33] Darabian, H., Dehghantanha, A., Hashemi, S., Homayoun, S., & Choo, K. K. R. (2020). An opcode-based technique for polymorphic Internet of Things malware detection. Concurrency and Computation: Practice and Experience, 32, e5173.