Enhancing Mac OS Malware Detection through Machine Learning and Mach-O File Analysis

Andrew Thaeler*, Yagmur Yigit*, Leandros Maglaras*, William J Buchanan*,
Naghmeh Moradpoor*, and Gordon Russell*
* School of Computing, Engineering and The Build Environment,
Edinburgh Napier University, United Kingdom
Email: 40497204@live.napier.ac.uk, yagmur.yigit@napier.ac.uk, L.Maglaras@napier.ac.uk,
b.buchanan@napier.ac.uk N.Moradpoor@napier.ac.uk, G.Russell@napier.ac.uk

Abstract—**Malware research has predominantly focused on Windows and Android operating systems, leaving Mac OS malware relatively unexplored. This paper addresses the growing threat of Mac OS malware by leveraging Machine Learning (ML) techniques. We propose a novel system for Mac malware detection that extends beyond traditional executables to include various Mach-O file types. Our research encompasses feature selection, data sets, and the implementation of ML classifiers. We meticulously evaluate system performance using Precision, Recall, F1 score, and Accuracy metrics. Our findings highlight the challenges and opportunities in Mac malware detection and provide valuable insights for future research.**

Index Terms—**Mac OS Malware Detection, Mach-O Files, Malware Detection, Static Malware Analysis.**

I. Introduction

Malware detection has traditionally been focused on Microsoft Windows and Android operating systems due to their historical dominance in the desktop and mobile markets. Much of the research in this domain has centred on these platforms, leveraging Machine Learning (ML) algorithms such as Decision Trees and Support Vector Machines for malware detection. However, these models' choice of data features can vary significantly between different operating systems and file formats. Typically, ML models for malware detection have depended on static analysis features related to the Windows Portable Executable (PE) file format, Windows libraries, and Windows system [1].

Contrary to popular belief, Apple's Mac OS has emerged as the second most popular desktop operating system, with a market share estimated at 17 % in March 2023 [2]. In January 2023, Statista estimated Mac OS to hold a 15 % share of the desktop market [3]. Despite its popularity, Mac OS has faced security challenges, and some security researchers argue that it may be less secure than Windows [4], [5], [6]. The volume of Mac-specific malicious programs and potentially unwanted software, such as adware, has been rising, outpacing the growth rate of Windows-focused threats. A researcher at Malwarebytes reported that the rate of virus detections on Macs was more than double that on Windows devices in 2019 on a per-device basis [7]. Apple Computer's Senior Vice President of Software Engineering acknowledged the increasing challenge of Mac malware in 2021, emphasizing the need for improved security measures [8].

In the realm of ML, malware detection approaches can be broadly categorized as either supervised or unsupervised. In supervised learning, the training dataset includes input and corresponding output labels. Conversely, unsupervised learning operates solely on input data, typically used for clustering similar objects. Previous research has demonstrated that supervised learning outperforms unsupervised methods for malware detection [9]. This paper focuses on supervised learning, utilizing known malware samples for detection. Additionally, this research focuses on static analysis, which involves the examination of files stored on disk without executing their code. Given the challenges and risks of executing malware for machine learning purposes, static analysis is a safer and more practical approach. Static analysis of disk-based files presents its own set of challenges, as attackers employ various techniques to evade detection [10] [11]. Recent advancements in cloud-based security and big data analytics have elevated static malware detection despite historical limitations [12]. Suspicious samples can now be analyzed centrally, improving the detection of malware files. However, attackers have also evolved their tactics, transitioning to fileless malware approaches [13]. This research aims to identify optimal features for malware detection using Machine Learning, specifically targeting Mac OS malware. The primary goal is to maximize the malware detection rate while minimizing false alarms. Previous works in this domain have predominantly focused on metadata and import tables of Mach-O executables, overlooking critical factors such as strings, entropy, and file size. The research will involve a trade-off analysis between the number of features, computation time, false positive rates, underfitting risks, and overfitting risks.

The rest of the paper is organized as follows. Section II surveys related studies in the literature. The proposed solution and the performance evaluation are explained in Section III and Section IV, respectively. We conclude the paper in Section V.

II. Related Work

In this section, we review notable contributions from the existing literature, shedding light on the methodologies and insights gained from previous work. Schultz *et al.* laid a foundation for this research by proposing a data mining approach to identify malware [14]. Their work focused on Windows PE and achieved remarkable results by employing features related to loaded Dynamically Linked Libraries (DLL), functions called from DLLs, and the frequency of function calls. This pioneering research set the stage for subsequent investigations in the field. Ucci *et al.* conducted a comprehensive survey of sixty-four papers dedicated to ML for detecting malicious Windows executables [15]. Their survey highlighted key findings, including the challenges posed by obfuscation, packing, and encryption in static analysis, the effectiveness of opcodes, Application Program Interfaces (APIs), and System Calls as significant features, and the imbalance problem in dataset composition. The authors emphasized the complexity of handling imbalanced datasets in malware detection due to the prevalence of non-malware instances. Another research noted the importance of functions in the IAT, offering valuable insights into detecting malware [16]. However, attackers can exploit compiler-generated functions, making feature selection and data analysis complex.

In the realm of Mac OS malware detection, research has been relatively limited compared to Windows-based studies [17]. However, a work contributed to early research on Mac malware detection using ML [18]. This approach involved feature extraction from Mach-O executable headers and load commands, focusing on metadata and structural features. Rotation Forest and Random Forest algorithms proved effective for this task, marking an early success in Mac malware detection through ML. Subsequent researchers built upon this foundation by exploring features specific to Mac OS malware detection. Pajouh *et al.* utilized metadata features from Mach-O executables and introduced features related to DLLs and their probabilities [5]. Their approach addressed the data imbalance issue using the Synthetic Minority Oversampling Technique (SMOTE) but noted increased false positive rates. The research underscored the relevance of library features in classification. Sahoo *et al.* emphasized the significance of the number of dynamically loaded libraries (`LoadDYLIB`) [19], while Gharghasheh *et al.* favoured ensemble classifiers for enhanced performance [20]. Chen *et al.* furthered the exploration of Mac malware detection, emphasizing the importance of Decision Trees in achieving high accuracy [21]. Their research retained metadata features and incorporated individual features for loaded DLLs, highlighting various approaches within this research domain.

Moreover, n-gram features, which involve sequences of objects such as instructions, bytes, file loads, system calls, and string characters, have been widely explored. However, their effectiveness is debated, with some researchers suggesting that obfuscation can render them less reliable. Overfitting and diminishing returns with larger n-grams have been observed [22]. Strings have consistently shown promise as features for malware detection, with their presence or absence serving as valuable indicators. The issue of handling large numbers of strings has been addressed through hashing techniques, reducing feature dimensionality while preserving relevancy [23]. These insights from existing literature provide a solid foundation for our research endeavours, guiding our pursuit of enhanced Mac OS malware detection through effective machine learning models.

III. Proposed System Model

In this section, we present the design and implementation of our proposed system for detecting Mac malware. Our approach goes beyond traditional Mac malware detection, encompassing various types of Mach-O files, including executables, libraries, object code, and core dumps. Previous research primarily focused on Mach-O executables, but our system aims to address supply chain attacks that modify system files and libraries, expanding the scope of detection. For instance, the SeaFlower malware employed a modified dynamic library to steal information [24], while the XCodeGhost malware altered object files within Apple's XCode development environment to inject malicious code [25]. To achieve this broader coverage, we analyzed all Mach-O files identified by their magic bytes, which include `0xCAFEBABE`, `0xCFFAEDFE`, `0xCEFAEDFE`, `0xFEEDFACE`, `0xFEEDFACF`, or `0xBEBAFECA`.

A. Data Sets

We collected malware samples from various sources to train and evaluate our Mac malware detection system. These sources include:

1) Patrick Warder's Objective See data set, curated over eight years and widely recognized in the research community [26].
2) MalwareSamples provided samples uploaded in February 2021 and used in prior research [27].
3) The Contagio data set, published by Mila Parkour in 2013, with additional samples over the years [28].
4) MalwareBazaar from Abuse.ch, a live repository focusing on recent samples [29].

We excluded the VirusShare repository due to its limited number of Mach-O executables, and three malware samples (Electrum, ElectrumStealer, and InstallCore) were excluded due to archive/zip errors.

After addressing duplicates and verifying the samples using McAfee and VirusTotal, our consolidated malware test set included 852 samples. We also set aside 47 samples as a validation set.

Table I provides an overview of the sample distribution among different repositories and architectures.

B. Feature Selection

Feature selection is a crucial aspect of our Mac malware detection system. We initially considered using n-grams for feature extraction but opted for a more effective approach based on strings. Strings offer several advantages: simplicity, efficient parsing, and resilience against exponential feature growth. To extract strings, we utilized the GNU Binutils 2.4 strings utility.

Our feature selection process involved identifying suspicious strings that were prevalent in malware but rare in goodware. We began with a set of 984 suspicious strings,

TABLE I
FILTERED DATASET AFTER REMOVING DUPLICATE MALWARE AND VERIFICATION USING MCAFEE/VIRUSTOTAL

Repository	Total	Universal Binary	64 Bit	32 Bit
Objective See	239	52	154	33
Malware Samples	479	15	429	35
Contagio	101	16	33	52
MalwareBazaar	94	17	77	0
Consolidated Malware Test Set	852	85	640	127
Malware Validation Set	47	8	39	0
Goodware x64	17968	15561	2359	48
Goodware M2	14365	12206	2092	67

which appeared in more than 10 % of malware samples and less than 5 % of goodware samples. However, after initial testing, we revised our feature set to include 1693 suspicious strings.

Additionally, we considered metadata features extracted from Mach-O files. These included entropy, the number of load commands (nlcs), the size of load commands (slcs), cputype, subtype, filetype, flags, and file size. We normalized these features and used them as part of our feature set.

To capture the relative presence or absence of suspicious strings in a sample, we introduced a calculated feature called "ratio". This feature represents the number of suspicious strings found in a file divided by the maximum number of suspicious strings, which is 1693.

Table II provides an overview of the average values for selected features, highlighting the differences between malware and goodware.

TABLE II
THE AVERAGE VALUES OF MACH-O FILES

Feature	Goodware Average	Malware Average
Entropy	4.338	5.599
Suspicious Strings (of 984)	22.2	137.8
Suspicious Strings (of 1693)	19.7	157.0
File Size	2.3MB	2.1MB
Number of Load Commands	27.35	23.34
Size of Load Commands	3141	2926

C. Implementation

Our system for Mac malware detection was implemented on an Ubuntu 64-bit Arm Server (version 22.04.2). We collected goodware samples from an M2 Max MacBook Pro and an Intel Core i9 MacBook Pro, running Mac OS Ventura 13.5. These laptops were in typical student configurations with various software applications and development tools.

We utilized Python 3.11.4, Scikit-Learn 1.3.0, and Numpy-1.25.2 for data analysis on an M2 Max MacBook Pro. Our machine learning algorithms included nine different classifiers: K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), Stochastic Gradient Descent (SGD), Naive Bayes (NB), Gradient Boosting (GB), Multi-Layer Perceptron (MLP), Support Vector Machine (SVM), and Bagging Classifier (BAG).

These classifiers were trained and evaluated using 5-fold cross-validation on our dataset. We also conducted multiple runs for some classifiers to account for randomness and averaged the results.

D. Metrics

We assessed the performance of our Mac malware detection system using several metrics, including Precision, Recall, F1 score, and Accuracy. The equation of the metrics can be seen in Equations 1, 2, 3, and 4, respectively. We employ the following variables in the equations: TP to denote true positives, TN to signify true negatives, FP to represent false positives, and FN to stand for false negatives. The F1 score encompasses Precision and Recall, offering a harmonious performance metric. Precision measures the rate of false positives, while Recall measures the rate of true positives. Validation Set Accuracy indicates the system's performance on "newer" malware samples.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

To further illustrate the trade-offs between true positive rates and false positive rates for our RF classifier, we created Receiver Operating Characteristic (ROC) curves, as can be seen in Fig 1 and Fig 2. Our proposed system combines carefully chosen characteristics and machine learning classifiers to detect Mac malware. By analyzing various types of Mach-O files and utilizing a diverse dataset, we aim to enhance the detection of Mac malware, including supply chain attacks that target system files and libraries. Our system's performance is rigorously evaluated using various metrics and cross-validation techniques.

IV. PERFORMANCE EVALUATION

In this section, we delve into the performance assessment of our malware detection models. We meticulously evaluate their proficiency in identifying malicious software, specifically emphasising their capability to detect previously undiscovered malware, as exemplified by the validation dataset. It is crucial to highlight that our detection systems operated without any prior awareness of the validation malware, all of which was sourced within the past 18 months. To gauge the effectiveness of our models, we conduct a comparative analysis against McAfee, a widely used commercial antivirus solution.

We began our evaluation by testing the KNN classifier with various numbers of neighbours ranging from 1 to 10. The KNN models exhibited moderate F1 scores, but their validation accuracy on new malware averaged only 43 % for M2 goodware and 36 % for x64 goodware. These subpar

TABLE III
THE PERFORMANCE RESULTS FOR M2 MAC MODEL

Features	DT	SGD	RF	MLP	GB	SVM	BAG
TN	14138.6	14187.3	14279.1	14219.1	14230.0	14223	14240.3
FP	226.4	177.7	85.9	145.9	135.0	142	124.7
FN	70.5	103.5	97.5	92.0	106.6	212	81.6
TP	781.5	748.5	754.5	760.0	745.4	640	770.4
Precision	0.7754	0.8081	0.8978	0.8389	0.8467	0.8184	0.8607
Recall	0.9172	0.8785	0.8856	0.8920	0.8749	0.7511	0.9042
F1 Score	0.8404	0.8419	0.8916	0.8647	0.8605	0.7883	0.8819
Validation Accuracy	0.6745	0.6191	0.6191	0.6447	0.5319	0.7021	0.6213

TABLE IV
THE PERFORMANCE RESULTS FOR X64 MAC MODEL

Features	DT	SGD	RF	MLP	GB	SVM	BAG
TN	17455.4	17745.2	17854.3	17735.1	17723.5	17728	17631.4
FP	512.6	222.8	113.7	232.9	244.5	240	336.6
FN	130.3	135.1	170.7	109.8	184.4	194	153.6
TP	721.7	716.9	681.3	742.2	667.6	658	698.4
Precision	0.5847	0.7629	0.8570	0.7612	0.7319	0.7327	0.6748
Recall	0.8471	0.8414	0.7996	0.8711	0.7835	0.7723	0.8197
F1 Score	0.6918	0.8002	0.8273	0.8124	0.7569	0.7520	0.7402
Validation Accuracy	0.7085	0.6213	0.6574	0.6319	0.6170	0.6808	0.6128

TABLE V
IMPACT OF FEATURES ON F1 SCORE AND VALIDATION PERFORMANCE
(RESULTS BASED ON THE M2 MAC DATASET)

Feature	984 Strings	1693 Strings	SSE	SSE+2	SSE+6	SSE+6 +Ratio
F1	0.8140	0.8582	0.8688	0.8710	0.8912	0.8916
Validation	0.6191	0.5913	0.6978	0.6894	0.6894	0.6191

results can be attributed to the class imbalance, with a ratio of 16.8:1 for M2 and 21:1 for x64, as KNN's performance tends to degrade under such conditions. Additionally, due to the high dimensionality of our feature space and its sparsity, finding the nearest neighbours became challenging, a phenomenon known as the "curse of dimensionality." The NB Gaussian classifier performed poorly, which was expected due to its known limitations with the class imbalance and feature independence assumption, which did not hold true for our dataset.

We also conducted tests using smaller subsets of goodware samples (1000, 3000, 4000 samples), which showed promising accuracy against new and test-set malware. However, these results did not accurately replicate the challenge of false positives that arise from a large, realistic dataset with noise. Furthermore, we observed that the F1 scores for M2 Mac models consistently outperformed those for x64 Mac models. This discrepancy might be influenced by the larger quantity of x64 goodware, potentially biasing the models towards goodware. Since Macs generally employed an Intel-based architecture until 2021, most malware is created for the Intel architecture, which raises the possibility of another explanation relating to the Instruction Set Architecture. Less than 11 % of the malware in the training set used ARM64 instructions, indicating that M2 ARM-based goodware is more recognisable from earlier malware than x64 Intel-based goodware.

Using two separate goodware datasets and the malware set,

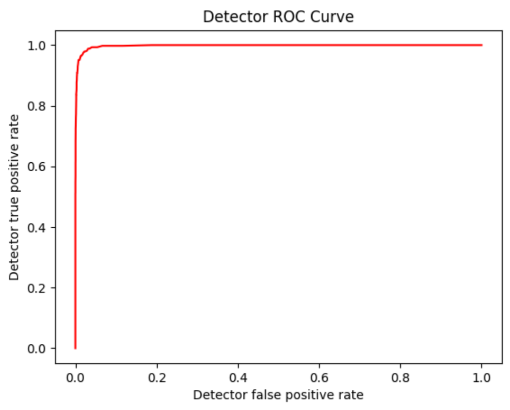

Fig. 1. Receiver Operating Characteristic Random Forest x64 *(17968 Goodware - 852 Malware)*

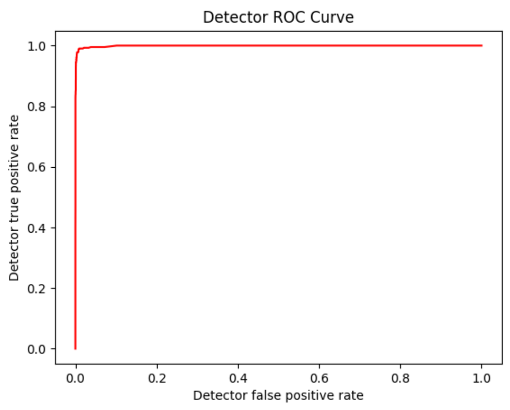

Fig. 2. ROC Random Forest M2 *(14279 Goodware - 852 Malware)*

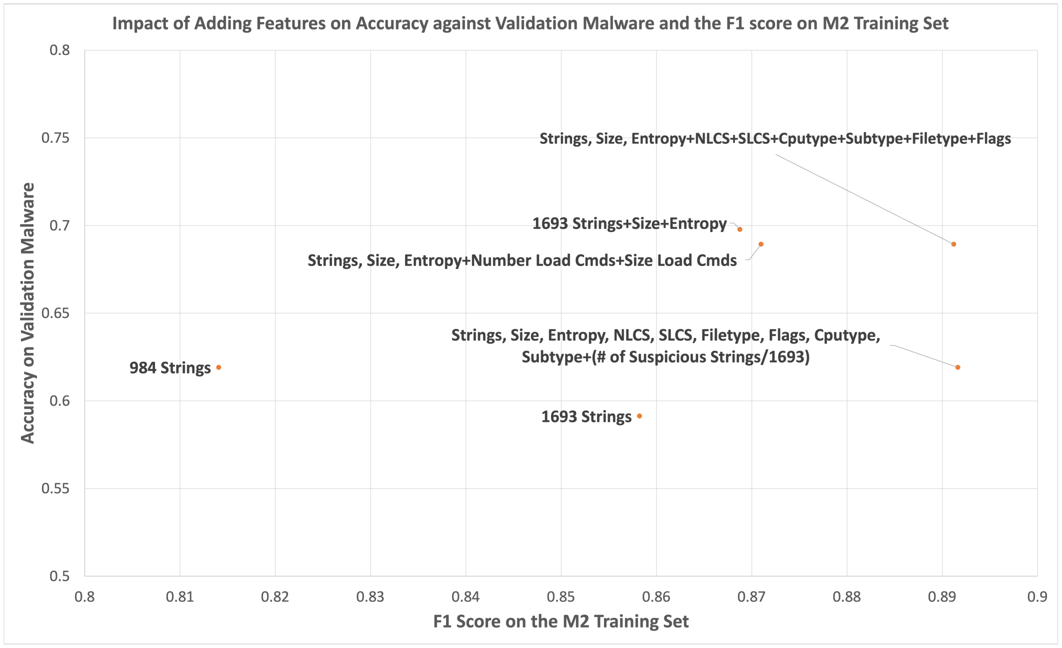

Fig. 3. Impact of added features on F1 score and validation malware detection. *(Analysis conducted using the M2 dataset.)*

TABLE VI
TOP 25 FEATURES MOST FREQUENTLY OCCURRING IN 20 DECISION TREES

Feature	Appearances in 20 Trees	Remarks
entropy	207	external
filesize	168	external
nlcs	92	metadata
ratio	57	metadata
cputype	57	metadata
slcs	55	metadata
flags	52	metadata
__program_vars	51	string
filetype	44	metadata
1N0-	40	
/Users/	34	directory path
/usr/lib/libgcc_s.1.dylib	30	gcc standard library
@_system	26	system call
_system	21	system call
_setuid	21	set user id
__objc_nlclslist__DATA	21	objective-c class list
subtype	20	metadata
c—w{	20	
conn	20	?network related
com.zoenzo.iMyMac	20	malicious url
coin	20	?cryptocurrency
Reliance	20	
L$hL	20	
/usr/lib/libiconv.2.dylib	20	character set translator
/bin/bash	20	shell

we compared the performance of the nine machine-learning techniques. Table III and Table IV present an overview of the relative performance of these algorithms concerning F1 scores on the training data and the detection of validation malware. While a high F1 score on the training data is desirable, our primary goal is to detect previously unseen malware from the validation set. Many of the algorithms displayed similar performance levels, with detection rates of validation malware ranging from 60 % to 70 % and F1 scores in the 0.8 to 0.9 range. Based on this data, RF emerged as a strong candidate due to its consistent performance between the two datasets, high F1 scores (indicating low false positives), and its ability to detect validation malware on par with other detectors.

We conducted an analysis of the feature impact on the detection models. Fig. 3 and Table V demonstrate the impact of adding various features on the F1 scores and validation accuracy using the M2 Mac dataset. We started with 984 suspicious string features and observed an increase in the F1 score when expanding the feature set to 1693. However, subsequent iterations, which included features like File Size, Entropy, nlcs, slcs, cputype, subtype, filetype, and flags, did not consistently result in improved detection of newer validation malware. Some features, such as cputype and subtype, performed well on the training malware set but caused false negatives when applied to the validation set, particularly for ARM64 malware. This underscores the challenges in static malware analysis posed by the evolution of hardware and software.

Table VI presents the 25 most popular features that appeared in the DTs. These features include entropy, filesize, ratio, and various internal metadata elements, all providing significant information gain for malware detection. Many strings associated with malware also appeared in the list. Interestingly, only 442 of the 1693 suspicious string features appeared in any of the 20 DTs, suggesting that some string features could be pruned from the datasets without affecting detector performance.

Our detectors performed competitively with commercial antivirus solutions when tested against historical malware. However, their performance dropped by 10 % - 15 % when faced with newer validation samples. To enhance their capabilities, our detectors would benefit from additional malware samples, especially those representing newer ARM64 malware. Furthermore, feature sets for malware detection should be dynamic, requiring continuous updates and tuning as both malware and goodware evolve over time. This comprehensive performance evaluation provides invaluable insights into the strengths and weaknesses of our developed models, contributing to the advancement of static malware analysis techniques.

V. Conclusion

This paper addresses the pressing need for effective Mac malware detection by developing a robust system that leverages Machine Learning. Our approach goes beyond conventional executables to include a diverse range of Mach-O file types, broadening the scope of detection to supply chain attacks. We extensively selected features and evaluated nine ML classifiers, revealing Random Forest as a promising candidate. Our results demonstrate competitive performance with commercial antivirus solutions against historical malware. However, the detection of newer validation samples poses challenges, necessitating continuous model updates and the inclusion of more recent malware samples, especially those targeting ARM64 architecture. We conclude that our research provides valuable insights into enhancing Mac malware detection and contributes to the evolution of static malware analysis techniques. Future work can explore reasons behind low detection rates for newer samples, incorporate more malware samples, fine-tune ML algorithms, address class imbalance, optimize features, and consider iOS devices. These directions promise to advance Mac OS malware detection and bolster security.

References

[1] J. Saxe and K. Berlin, "Deep Neural Network-based Malware Detection using Two-dimensional Binary Program Features," in *2015 10th international conference on malicious and unwanted software (MALWARE)*. IEEE, 2015, pp. 11–20.

[2] Statcounter, "Statcounter GlobalStats: Operating System Market Share Worldwide," https://gs.statcounter.com/os-market-share, 2023, Accessed: 2023-03-31.

[3] Statista, "Global Market Share held by Operating Systems for Desktop PCs, from January 2013 to January 2023," https://www.statista.com/statistics/218089/global-market-share-of-windows-7/, 2023, Accessed: 2023-03-31.

[4] M. Hypponen, *If it's smart, it's vulnerable*. Wiley, 2022.

[5] H. H. Pajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "Intelligent OS X Malware Threat Detection with Code Inspection," *Journal of Computer Virology and Hacking Techniques*, vol. 14, pp. 213–223, 2018.

[6] A. Robles-Durazno, N. Moradpoor, J. McWhinnie, G. Russell, and I. Maneru-Marin, "Implementation and detection of novel attacks to the plc memory of a clean water supply system," in *Technology Trends*. Cham: Springer International Publishing, 2019, pp. 91–103.

[7] T. Reed, "Mac Threat Detections on the Rise in 2019," https://www.malwarebytes.com/blog/news/2019/12/mac-threat-detections-on-the-rise-in-2019, 2019, Accessed: 2023-08-15.

[8] C. Welch, "The level of Mac Malware is not Acceptable, says Apple's Craig Federighi at Epic trial," *The Verge*, 2021, Accessed: 2023-04-01.

[9] K. O. Babaagba and S. O. Adesanya, "A Study on the Effect of Feature Selection on Malware Analysis using Machine Learning," in *Proceedings of the 2019 8th international conference on educational and information technology*, 2019, pp. 51–55.

[10] D. Holmes, M. Papathanasaki, L. Maglaras, M. A. Ferrag, S. Nepal, and H. Janicke, "Digital Twins and Cyber Security – solution or challenge?" in *2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, 2021, pp. 1–8.

[11] N. Yehoshua and U. Kosayev, *Antivirus Bypass Techniques: Learn practical techniques and tactics to combat, bypass, and evade antivirus software*. Packt Publishing Ltd, 2021.

[12] Y. Yigit, L. D. Nguyen, M. Ozdem, O. K. Kinaci, T. Hoang, B. Canberk, and T. Q. Duong, "Twinport: 5g drone-assisted data collection with digital twin for smart seaports," in *Scientific Reports*. Nature, 2023.

[13] Microsoft, "Volt Typhoon Targets US Critical Infrastructure with Living-off-the-land Techniques," https://www.microsoft.com/en-us/security/blog/2023/05/24/volt-typhoon-targets-us-critical-infrastructure-with-living-off-the-land-techniques/, 2023.

[14] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data Mining Methods for Detection of New Malicious Executables," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 2000, pp. 38–49.

[15] D. Ucci, L. Aniello, and R. Baldoni, "Survey of Machine Learning Techniques for Malware Analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[16] M. Lester, "Threat Hunting with Function Imports," https://practicalsecurityanalytics.com/threat-hunting-with-function-imports/, November 2019, Accessed: 2023-04-14.

[17] G. Secinti, P. B. Darian, B. Canberk, and K. R. Chowdhury, "Resilient end-to-end Connectivity for Software-defined Unmanned Aerial Vehicular Networks," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017, pp. 1–5.

[18] E. Walkup, "Mac Malware Detection via Static File Structure Analysis," https://cs229.stanford.edu/proj2014/, 2014, accessed: 2023-03-31.

[19] D. Sahoo and Y. Dhawan, "Evaluation of Supervised and Unsupervised Machine Learning Classifiers for Mac OS Malware Detection," *Handbook of Big Data Analytics and Forensics*, pp. 159–175, 2022.

[20] S. E. Gharghasheh and S. Hadayeghparast, "Mac OS X Malware Detection with Supervised Machine Learning Algorithms," *Handbook of Big Data Analytics and Forensics*, pp. 193–208, 2022.

[21] A. C. Chen and K. Wulff, "Machine Learning for OSX Malware Detection," *Handbook of Big Data Analytics and Forensics*, pp. 209–222, 2022.

[22] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, "An Investigation of byte n-gram Features for Malware Classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, pp. 1–20, 2018.

[23] J. Saxe and H. Sanders, *Malware Data Science: Attack Detection and Attribution*. No Starch Press, 2018.

[24] Medium, "How SeaFlower Installs Backdoors in iOS/Android Web3 Wallets to Steal Your Seed Phrase," https://blog.confiant.com/how-seaflower-installs-backdoors-in-ios-android-web3-wallets-to-steal-your-seed-phrase-d25f0ccdffce, 2022, Accessed: 2023-08-15.

[25] C. Xiao, "Malware XcodeGhost Infects 39 iOS Apps, Including WeChat, Affecting Hundreds of Millions of Users," https://unit42.paloaltonetworks.com/malware-xcodeghost-infects-39-ios-apps-including-wechat-affecting-hundreds-of-millions-of-users/, 2015, Accessed: 2023-08-15.

[26] Objective-See, "Mac Malware Collection, a Public Collection of Malicious Code Targeting MacOS," https://objective-see.org/malware.html, Accessed: 2023-04-01.

[27] MalwareSamples, "Mr. Malware MalwareSamples GitHub Repository," https://github.com/MalwareSamples/Macos-Malware-Samples, 2021.

[28] Contagio, "OSX Malware and Exploit Collection, Links and Resources for OSX Malware Analysis," https://contagiodump.blogspot.com/2013/11/osx-malware-and-exploit-collection-100.html, 2013.

[29] A. Ch, "MalwareBazaar Data Set," https://bazaar.abuse.ch/, 2023, Accessed: 2023-08-15.