

Characterizing Tactics, Techniques, and Procedures in the macOS Threat Landscape

Daniel Lastanao Miró^a, Javier Carrillo-Mondéjar^a, Ricardo J. Rodríguez^{b,*}

^aDpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain

^bInstituto de Investigación en Ingeniería de Aragón (I3A), Universidad de Zaragoza, Spain

Abstract

As macOS systems increasingly become malware targets, understanding the tactics, techniques, and procedures (TTPs) used by adversaries is essential to improving defense strategies. This paper provides a systematic and detailed analysis of macOS malware using the MITRE ATT&CK framework, focusing on TTPs at key stages of the malware attack cycle. Leveraging a comprehensive dataset of 57,636 macOS malware samples collected between November 2006 and October 2024, we employ both static and dynamic analysis techniques to uncover patterns in adversary behavior. Our analysis, primarily based on static analysis techniques, offers a broad representation of macOS malware and highlights common characteristics across samples. While we only partially explore dynamic behaviors, we identify recurring patterns that align with specific TTPs in the MITRE ATT&CK framework, such as persistence and defense evasion. This mapping contributes to a more structured understanding of macOS threats and can help inform future detection and mitigation efforts.

Keywords: macOS malware, MITRE ATT&CK framework, Malware behavior, Static and dynamic analysis

1. Introduction

In recent years, the macOS malware threat landscape has expanded rapidly, with significant growth in both the volume and sophistication of attacks targeting macOS systems, according to current anti-malware industry reports (Intel471, 2024; Malwarebytes, 2024). Current data highlights an upward trend in macOS-focused threats, driven by increased adoption of macOS devices in both personal and corporate environments (Kandji, 2024). As a result, the macOS platform, once perceived as relatively secure, is now an increasing target for adversaries looking to exploit its vulnerabilities.

Understanding the malware lifecycle, which spans multiple stages, from initial access to execution, persistence, and evasion, and how adversaries operate is essential to effectively countering these threats. In this sense, the MITRE ATT&CK framework (MITRE Corporation, 2024) provides a systematic and comprehensive method for mapping adversary behavior across these stages. By organizing known attack patterns into Tactics, Techniques, and Procedures (TTPs), the framework facilitates a granular understanding of malicious operations. For instance, a representative TTP of the *persistence* tactic is the use of *launch agents* (technique T1543.001), where the malware creates a *.plist file in ~/Library/LaunchAgents to ensure its execution upon user login (procedure). This allows adversaries to maintain their presence on the system over time.

Applying this framework to the study of macOS malware improves detection and mitigation strategies by identifying behavioral patterns. It not only facilitates the consistent classification of malicious activities, but also highlights potential intervention points throughout the attack lifecycle. Ultimately, MITRE ATT&CK contributes to building a more structured, threat-based defense strategy against macOS-specific adversaries.

Despite the growing threat landscape, research focused on macOS malware analysis (Wardle, 2022) remains limited compared to that for Windows (Oosthoek and Doerr, 2019; Mousaileb et al., 2021; Tekiner et al., 2021; Ugarte-Pedrero et al., 2019; Botacin et al., 2021a), Linux (Cozzi et al., 2018; Alrawi et al., 2021; Al Alsadi et al., 2022), or Android (Acar et al., 2016; Qamar et al., 2019; Qiu et al., 2020). This gap in knowledge restricts our understanding of macOS-specific malware behaviors and how effectively current defenses can detect and mitigate them. This paper aims to close this gap, addressing this limitation to improve the state of macOS system security.

In this paper, we present an in-depth analysis of macOS malware using static and dynamic analysis techniques (Wardle, 2022; Zeller, 2003), framed in the context of the malware attack lifecycle using the MITRE ATT&CK framework. Specifically, we collected macOS malware samples from multiple sources, including VirusTotal (vir, 2024), VirusShare (Corvus Forensics, 2024), and MalwareBazaar (mal, 2024), covering the period from November 2006 to October 2024. Static analysis involves inspecting binaries without executing them, focusing on characteristics such as file structures or embedded metadata (Yong Wong et al., 2021). In contrast, dynamic analysis observes the behavior of binaries during execution in a controlled environment, capturing interactions, system state changes, and network activity. For static analysis, we use tools

*Corresponding author

Email addresses: dlastanao@unizar.es (Daniel Lastanao Miró), jcarrillo@unizar.es (Javier Carrillo-Mondéjar), ricardo@unizar.es (Ricardo J. Rodríguez)

URL: <https://webdiis.unizar.es/~ricardo/> (Ricardo J. Rodríguez)

such as radare2 (Radare2 Team, 2017), AVClass (Sebastián and Caballero, 2020) and Detect It Easy (DIE) (die, 2022), along with metadata from VirusTotal (vir, 2024). For dynamic analysis, we use VirusTotal’s macOS-focused sandbox environments to observe runtime behaviors, including executed commands, file interactions, and MITRE ATT&CK mappings.

Contributions. In this work, we make the following main contributions:

- (i) *Extensive dataset and methodology for static and dynamic analysis of macOS malware.* We present a large-scale dataset of 57,636 macOS malware samples, all of which were statically analyzed. From this set, we provide detailed information derived from 21,967 (38.11%) samples that underwent dynamic behavioral analysis. This constitutes one of the largest publicly documented collections of macOS malware to date, providing a strong empirical basis for systematic study. We make this dataset available to foster further research and benchmarking, laying the foundation for future advancements in macOS malware analysis (Daniel Lastanao Miró, 2025). Furthermore, we detail the complete analytical pipeline, including the tools, scripts, configurations, techniques used, to ensure full reproducibility and transparency, thus reinforcing our commitment to open science and replicable malware research.
- (ii) *Systematization of knowledge of the macOS malware landscape.* We conduct the first large-scale, structured systematization of macOS malware behaviors, organized around the different phases of the malware lifecycle. Using the MITRE ATT&CK framework as a reference, we identify recurring tactics, techniques, and procedures (TTPs) and reveal common adversary patterns specific to the macOS ecosystem.
- (iii) *Insights into malware behavior and attack patterns.* Based on our systematization, we extract and analyze empirical trends in the TTPs used by macOS malware. Our findings reveal the most frequent behavioral patterns throughout the malware lifecycle, such as extensive command and control activity, widespread use of defense evasion strategies, and little evidence of lateral movement. These results provide a data-driven characterization of the current macOS threat landscape and offer valuable insights into how adversaries prioritize persistence, stealth, and control when operating in this environment.

Organization. The rest of this paper is organized as follows. Section 2 provides a background on the MITRE ATT&CK framework and the Mach-O executable format, and macOS security mechanisms. Section 3 details our methodology for static and dynamic analysis. Section 4 describes the dataset used in our study, while Section 5 discusses the analysis of results. Section 6 provides a brief discussion of how malware differs across different operating systems, while Section 7 discusses the limitations of our work. Section 8 reviews related work on macOS malware. Finally, Section 9 concludes the paper and describes future work.

2. Background

To provide a basic understanding of the concepts discussed in this paper, we present background information, focusing on the MITRE ATT&CK framework, the Mach-O executable file format, and key security features of macOS.

2.1. MITRE ATT&CK Framework and macOS

The MITRE ATT&CK framework (MITRE Corporation, 2024) is a widely adopted knowledge base that provides a structured way to understanding and analyzing malicious actor behavior at different stages of an attack. It breaks down adversary actions into TTPs, serving as a valuable resource for identifying, categorizing, and mitigating threats.

In the context of macOS, MITRE ATT&CK provides a dedicated matrix with tactics and techniques specific to macOS systems, allowing security professionals to understand the nuances of macOS attacks and develop effective defense strategies. This matrix is particularly valuable in malware research, as it helps security teams identify gaps in their detection and defense capabilities. By mapping observed malware behavior to the ATT&CK matrix, threat analysts can determine which TTPs are currently monitored and which require enhanced detection coverage. For instance, if a malware sample is found to use the *Boot or Logon Autostart Execution* technique, security teams can assess their existing detection measures to ensure that attempts to automatically run a program during system startup or account login are properly logged and flagged.

As the time of writing, the MITRE ATT&CK matrix for macOS includes 12 tactics, 170 techniques, and 243 sub-techniques. *Tactics* represent the adversary’s general goals (i.e., the “why” behind each step of an attack), *techniques* describe how those goals are achieved (i.e., which specific methods are used to accomplish tactics), and *sub-techniques* provide more specific details about the methods used. *Procedures* refer to the exact steps or commands used to implement these techniques. These elements can be combined into TTP chains, which represent the sequential actions that adversaries take to achieve their goals. Each TTP is assigned a unique ID (for instance, T1547 for *Boot or Logon Autostart Execution*), facilitating streamlined automation and analysis.

2.2. The Mach-O Executable File Format

The Mach-O (*Mach Object*) format is the native executable file format for macOS and iOS (Levin, 2017). Introduced with NeXTSTEP and later adopted by macOS, it has become the preferred format for distributing binaries in these environments. The Mach-O format consists of multiple segments, each containing one or more sections that organize specific types of data or code. Unlike the Windows PE format (Microsoft, 2022), where headers are used to define executable characteristics, Mach-O segments determine how code and data are mapped into memory. Segments begin on a page boundary, but the sections within a segment do not necessarily align to page boundaries. This structure allows the Mach-O binary to be efficiently mapped into memory during the loading process.

A Mach-O binary is organized into three parts: the *header*, *load commands*, and *segments* (which contain sections) (Levin, 2017). The header begins with a magic value (typically `0xfeedface` for 32-bit little-endian binaries or `0xfeedfacf` for 64-bit little-endian binaries) and defines key attributes such as the object type, target architecture, flags, and number of load commands. The header object type defines whether the file is an executable, a dynamic library (dylib), a bundle, or another supported format. Load commands provide information about the rest of the object file, including segments and sections, which are typically listed in the order they are allocated in memory.

Segments are contiguous memory regions that are allocated from the object file during loading. Each segment has specific memory protections, as specified by its load command, and typically retains the same order and offsets as in the file, although this is not strictly required. Segments are further divided into *sections*, which are smaller subdivisions within a segment. Sections inherit memory protections from their parent segments, but contain different types of content, such as executable code, string literals, or symbol fragments, making segmentation necessary for efficient organization. For instance, the `__TEXT` segment contains read-only executable code and constant data, which can be shared among multiple processes to reduce memory usage. In contrast, the `__DATA` segment contains writable non-constant data, including sections for uninitialized variables and initialized global variables.

To accommodate the transition from PowerPC to Intel architectures in macOS 10.4, Apple introduced *Universal Binaries* (informally known as *fat binaries*) (Apple Support), which package multiple versions of a binary (each for a different architecture) into a single file. This allows for seamless execution on different CPU architectures, simplifying the distribution of binaries for different systems. When a Universal Binary is loaded, the kernel selects the appropriate architecture slice based on the current system, optimizing memory usage. Support for 32-bit Intel binaries began to be phased out with macOS 10.8, although compatibility continued through macOS 14 (Levin, 2017).

2.3. macOS Security Overview

macOS offers a unique set of security features designed to protect both the system and user data while maintaining a seamless user experience. macOS security begins with a layered architecture that includes hardware-based and software-based protections. At the hardware level, features such as Secure Enclave (Apple Support, 2024c) provide isolated processing for sensitive operations such as encryption, authentication, and key management. Additionally, the Secure Boot (Apple Support, 2021) process ensures that only trusted software is loaded during startup, effectively preventing malware from injecting itself into the system boot sequence.

On the software side, macOS includes System Integrity Protection (SIP) (Apple Support, 2023), which restricts the root user from modifying critical system files and folders. This dramatically reduces the attack surface for malware that attempts to escalate privileges by altering protected system components. SIP ensures that even if a user or malicious software gains root

access, crucial system files and directories remain protected. Similarly, Gatekeeper prevents untrusted and potentially harmful apps from running on the system by verifying the developer’s identity through a trusted certificate authority.

macOS also features XProtect and Notarisation as built-in anti-malware defenses (Apple Support, 2024b). XProtect is a signature-based malware detection tool, based on YARA rules (Naik et al., 2020), that scans apps when they are first launched or downloaded, blocking any that match known malware signatures. Notarisation, on the other hand, is a malware scanning service provided by Apple. Developers who want to distribute apps for macOS outside the App Store submit their apps for scanning as part of the distribution process. Apple reviews the software for known malware, and if no issues are found, issues a Notarisation ticket. Developers then attach this ticket to their app, allowing Gatekeeper to verify and launch the app even when offline.

Another critical security mechanism is App Sandbox, which limits the scope of an application interaction with the system and user data (Apple Support, 2024a). Applications must declare their intended functionality through entitlements, allowing macOS to strictly control and isolate their activities. This sandbox approach effectively minimizes the damage that malware can cause, as malicious apps are typically restricted to their own container and cannot affect the broader system or access sensitive information without explicit permissions.

3. Methodology

In this section, we describe our methodology. We start by describing the dataset used in this work. Next, we present the static analysis performed on each malware sample, detailing the techniques and tools applied to extract structural information. Finally, we discuss the dynamic analysis approach, including the methods used to observe malware behavior and extract relevant behavioral data.

3.1. Dataset

To better understand the characteristics of malware targeting macOS, we collected samples from several data sources, specifically VirusShare (Corvus Forensics, 2024), MalwareBazaar (mal, 2024), and VirusTotal (vir, 2024).

VirusShare and MalwareBazaar are malware repositories that archive community-contributed real-world malware samples, spanning a variety of operating systems and architectures. We collected all samples tagged as Mach-O that were available in VirusShare as of September 4, 2024, and in MalwareBazaar as of July 25, 2024. In total, these collections yielded 78,127 and 161 samples, respectively, targeting operating systems within the Apple ecosystem.

VirusTotal (VT) is a malware analysis and sharing platform used by hundreds of commercial security companies and thousands of researchers worldwide. VT offers a commercial service (called VT Intelligence) that allows users to search for samples that meet specific criteria that have been uploaded in the past 90 days. We leveraged this service to gather all Mach-O

samples uploaded to VT between April 26, 2024, and September 10, 2024, resulting in 8,803 samples. Next, we obtained VT’s analysis reports for each collected sample and discarded those that were not flagged as malware by at least five antivirus engines, following best practices in malware research (Zhu et al., 2020; Wang et al., 2023). This decision was made to ensure that all samples in the dataset are indeed malware and to minimize the risk of including false positives (Zhu et al., 2020).

After assembling the initial dataset, we performed preprocessing to remove non-Mach-O files, focusing exclusively on binaries specific to the macOS platform. Table 1 summarizes the distribution of Multipurpose Internet Mail Extensions (MIME) file types identified in the original dataset. MIME types describe the format and nature of a file based on its content, providing necessary information on how systems should interpret and handle them. This filtering only retained files with the MIME type `application/x-mach-binary`, identified using the `file` command, which classifies files based on their actual content rather than solely based on their extensions.

Among the retained Mach-O binaries, several file types were observed: `Executable` files are standard application binaries; `Dylib` files are dynamic libraries and framework components; `Bundle` files represent plugin modules; `Object` files are relocatable intermediate outputs generated by `gcc -c`, also used for deprecated 32-bit kernel extensions; `Kext_Bundle` files are kernel extensions; and `Dsym` files are debug symbol `.dsym` files generated by `gcc -g` (Levin, 2017).

Surprisingly, we found 36 files from the original dataset that were neither macOS nor iOS. These included compressed archives, Python scripts, Microsoft Word documents, and other unrelated formats, which were subsequently discarded. After this MIME type filtering, the resulting dataset consists of 65,716 Mach-O files. We then excluded 3,694 samples that specifically targeted Apple mobile devices, as these are outside the scope of our analysis. To ensure consistency, we verified the file type of each remaining sample using `radare2`. Only those explicitly classified as executables (capable of running directly on macOS) were retained. As a result, the final dataset consists of 57,636 Mach-O executable samples targeting Apple desktop systems.

We used `AVClass` (Sebastián and Caballero, 2020) to assign labels to the malware samples. This tool is widely adopted by the research community to classify malware samples, leveraging antivirus labels from VT reports to label a given sample. Using `AVClass`, we obtained labels for a total of 42,963 malware samples in 492 distinct families. In addition, we employed `AVClass` to identify Potentially Unwanted Programs (PUPs) within the dataset, detecting a total of 38,978 PUP files.

3.2. Static Analysis

We performed a comprehensive static analysis of all samples in our dataset to extract relevant metadata and better understand malware targeting macOS. This phase involved extracting various features from the binaries, each of which contributed to a multifaceted profile of the malware.

First, we analyzed the symbols present in the binaries, focusing on external libraries and functions imports. These symbols

Table 1: File MIME type distribution of the initial dataset.

File MIME Type	Count
	<code>Executable</code> (57,636)
	<code>Dylib</code> (3,624)
	<code>Bundle</code> (576)
<code>application/x-mach-binary</code> (macOS)	<code>Object</code> (150)
	<code>Kext_Bundle</code> (29)
	<code>Dsym</code> (7)
<code>application/x-mach-binary</code> (iOS)	3,694
<code>application/zip</code>	7
<code>application/x-xar</code>	10
<code>application/octet-stream</code>	1
<code>application/x-xz</code>	2
<code>application/zlib</code>	8
<code>application/x-cpio</code>	2
<code>application/msword</code>	1
<code>inode/x-empty</code>	1
<code>text/plain</code>	1
<code>application/java-archive</code>	1
<code>text/x-script.python</code>	2
Total	65,752

provide valuable insight into the malware’s dependencies and runtime behavior. Figure 1 presents the top 50 most frequently imported functions across all samples, revealing recurring patterns, such as a strong reliance on lazy binding, stack protection compilation, and dynamic memory. Next, we calculated the cyclomatic complexity of each binary to obtain a quantitative measure of the program’s logical complexity. This metric offers insight into the structural complexity and sophistication of the malware (Calleja et al., 2019). We also extracted the target architecture (e.g., `x86_64`, `arm64`) from each sample to understand platform-level trends. Additionally, we identified the programming language used in each binary by analyzing language-specific patterns and structures within the compiled code. Another important feature we examined was whether the binaries had been stripped of debugging symbols. The presence or absence of such symbols may indicate the malware authors’ intent to hinder reverse engineering and analysis efforts.

To extract these features, we employed `radare2` (Radare2 Team, 2017), a widely used open-source reverse engineering framework that supports programmable automation using `r2pipe`. This allowed us to perform large-scale analysis programmatically and consistently across all samples.

We also determined whether each sample is packed. Initially, we evaluated the entropy of the binary to identify potential packing, followed by applying a set of rules to detect the specific packer employed. The entropy of a binary measures the randomness of its byte structure; high entropy often indicates that the binary is packed or contains encrypted sections. For this detection process, we use the cross-platform tool `DIE` (die, 2022), which uses predefined signatures to identify a wide range of packers and compilers.

3.3. Dynamic Analysis

In our work, we also perform a dynamic analysis of the samples to understand the behavior of malware targeting macOS.

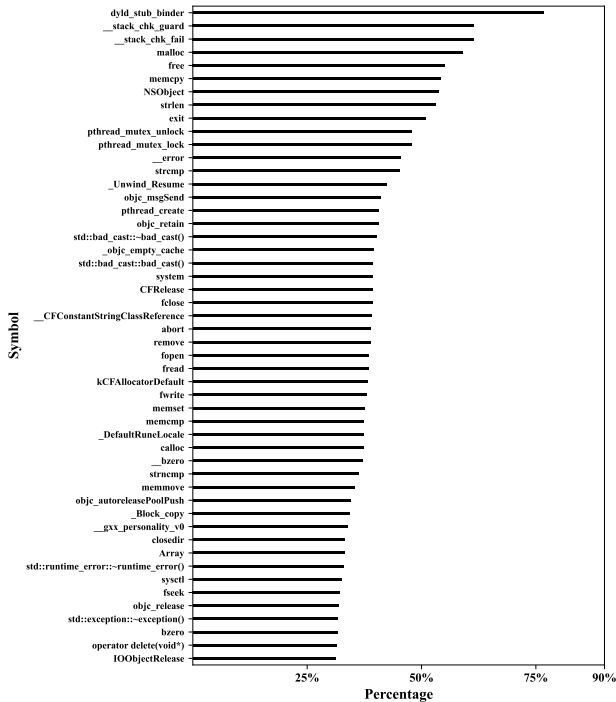


Figure 1: Top 50 function imports in the dataset.

To extract behavioral insights, we rely on VT. VT performs dynamic analysis using various sandbox environments provided by both its own infrastructure and third-party vendors, allowing for monitoring malware behavior during execution. For this work, we selected only the sandboxes operated directly by VT, given their greater reliability and the fact that none of the third-party sandboxes explicitly claim to support Mach-O binary analysis (VirusTotal, 2025). Specifically, VT offers three sandbox environments designed for analyzing Mach-O binaries: (i) *Box Of Apples*, which hooks system calls and supports x86 binaries; (ii) *OS X Sandbox*, running macOS 11.6, which incorporates MITRE ATT&CK signatures to map adversary behavior and supports x86 binaries (previously this sandbox supported ARM binaries); and (iii) *Zenbox macOS*, running macOS 13 on ARM, which also includes MITRE ATT&CK signatures and supports x86, x86_64, and ARM binaries. We acknowledge that relying on an external platform for dynamic analysis introduces certain limitations in our study, which are discussed in Section 7.

Using the information provided by VT’s reports, we extracted several essential elements for our analysis. These included mappings to the MITRE ATT&CK matrix, which offered insight into the tactics and techniques employed by the malware. We also collected details on process interactions, such as commands executed and files accessed, as well as network activity, including IP addresses, domains, and HTTP content types. Network data was further analyzed using NeutrinoAPI (NeutrinoAPI, 2025), which offers multiple services; in our case, we used the *IP Blocklist API*, which identifies potentially malicious or dangerous IP addresses, and the *Domain Lookup API*, which retrieves domain details and flags suspi-

cious or harmful domains. All domains were compared to DGArchive (DGArchive, 2025) a domain name database generated by domain generation algorithms (DGAs), commonly observed in malware campaigns. Finally, we gathered information about file interactions, focusing on files that were opened, created, or modified during execution.

However, like other dynamic analysis platforms, it is not always possible to collect data on malware execution due to limitations in the execution environment, such as OS version discrepancies, the absence of required external libraries, or the detection of a virtualized environment (Botacin et al., 2021b). As a result, only 21,967 included behavioral information, which represents approximately 38.11% of the total dataset. Of these, 15,469 are samples with MITRE ATT&CK data, which represents 26.83% of the overall dataset. For the remainder of this paper, all metrics and ratios concerning TTPs will be based on these 15,469 binaries. These samples span from December 2008 to October 2024.

4. Dataset Overview

As highlighted in Section 3.1, our final dataset consists of 57,636 Mach-O malware samples spread across different macOS-supported architectures. Table 2 provides a complete breakdown by architecture, detailing packed versus unpacked samples, samples with and without symbols, and samples with and without behavioral data. The majority of samples belong to the x86 64-bit architecture, accounting for 83.43% of the total dataset. This distribution aligns with expectations, given that Apple used Intel architectures from 2006 to 2020 before transitioning to ARM64 architectures (Apple Media Helpline, 2006, 2020).

Temporal Distribution of macOS Malware Samples. Figure 2 illustrates the temporal distribution of our dataset by CPU architecture, using VirusTotal’s “first seen” field to establish the chronology of each sample. This figure categorizes the samples by their target architecture and includes an additional group called *without behavior*, which represents binaries for which the VirusTotal sandbox did not record behavioral information. It is worth noting that all samples categorized into a specific CPU architecture group have associated behavioral information extracted from the sandbox reports. To complement this analysis, Figure 5 presents the same temporal distribution, broken down by CPU architecture across the various VirusTotal sandboxes, offering a more granular view on how different sandboxes report behavioral data over time.

The majority of samples originated in 2023 and 2024, reflecting a recent increase in macOS malware activity. Specifically, these two years account for 49.72% and 27.67% of the total dataset, respectively. While macOS malware has been around for many years (the earliest samples in our dataset date back to 2006), this trend supports recent findings that macOS has recently become a prime target for malware authors and cybercriminal groups (Intel471, 2024; Malwarebytes, 2024).

Statistical Analysis of macOS Malware Samples. We present statistical information on macOS-targeting malware to highlight distinctions within the majority of samples in our dataset.

Table 2: Distribution of the dataset.

Architecture	Count	Packed	Unpacked	Stripped	Non-stripped	With behavioral data	Without behavioral data
x86_64	48,083 (83.43%)	9,720 (20.22%)	38,363 (79.78%)	42,241 (87.88%)	5,842 (12.12%)	16,840 (35.03%)	31,243 (64.97%)
ARM_64	3,970 (6.89%)	634 (15.96%)	3,336 (84.04%)	2,922 (73.59%)	1,048 (26.41%)	1,089 (27.44%)	2,881 (72.56%)
x86_64, ARM_64	2,683 (4.66%)	86 (3.21%)	2,597 (96.79%)	1,646 (61.34%)	1,037 (38.66%)	1,971 (73.47%)	712 (26.53%)
x86_32	1,656 (2.87%)	90 (5.43%)	1,566 (94.57%)	1,510 (91.18%)	146 (8.82%)	1,284 (77.54%)	372 (22.46%)
x86_32, PPC_32	983 (1.71%)	6 (0.61%)	977 (99.39%)	982 (99.90%)	1 (0.10%)	736 (76.89%)	227 (23.11%)
PPC_32	159 (0.28%)	12 (7.55%)	147 (92.45%)	159 (100.00%)	0 (0.00%)	0 (0.00%)	159 (100.00%)
ARM_32	69 (0.12%)	20 (28.99%)	49 (71.01%)	69 (100.00%)	0 (0.00%)	0 (0.00%)	69 (100.00%)
x86_64, PPC_32	29 (0.05%)	0 (0.00%)	29 (100.00%)	28 (96.55%)	1 (3.45%)	23 (86.21%)	4 (13.79%)
PPC_64	2 (0.00%)	2 (100.00%)	0 (0.00%)	2 (100.00%)	0 (0.00%)	0 (0.00%)	2 (100.00%)
x86_64, PPC_64, PPC_32	2 (0.00%)	2 (100.00%)	0 (0.00%)	2 (100.00%)	0 (0.00%)	2 (100.00%)	0 (0.00%)
Total	57,636 (100.00%)	10,571 (18.34%)	47,065 (81.66%)	49,561 (85.99%)	8,075 (14.01%)	21,967 (38.13%)	35,669 (61.87%)

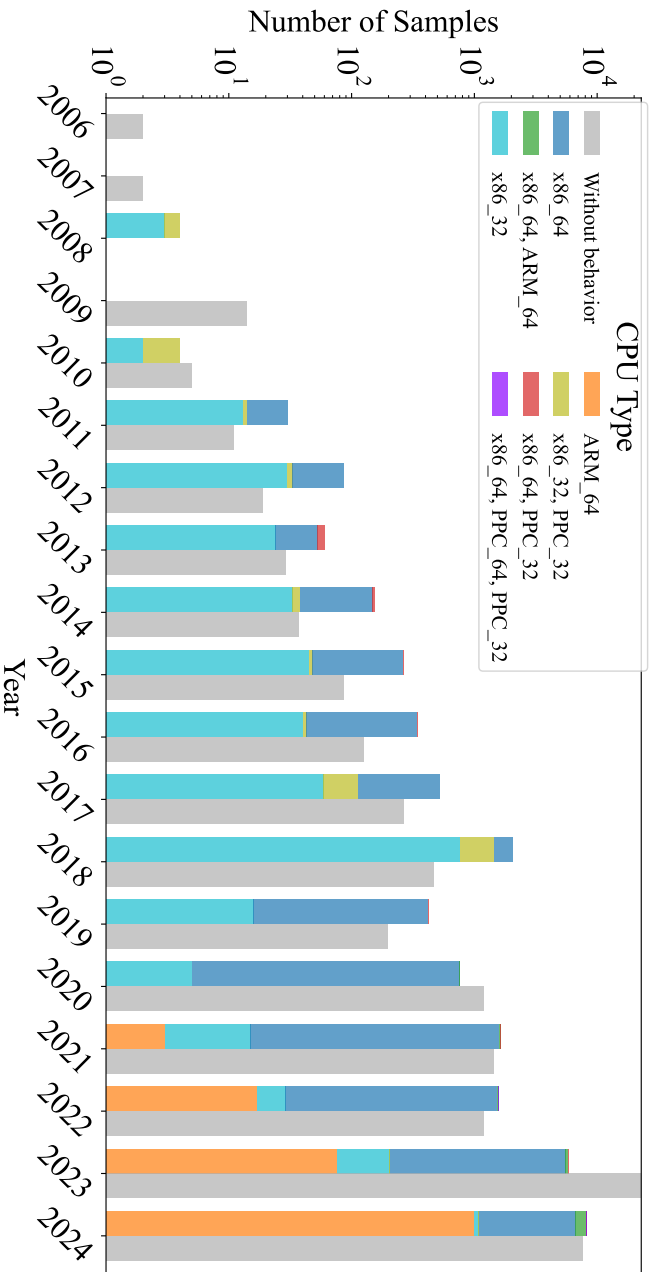


Figure 2: Annual distribution of malware samples by CPU type.

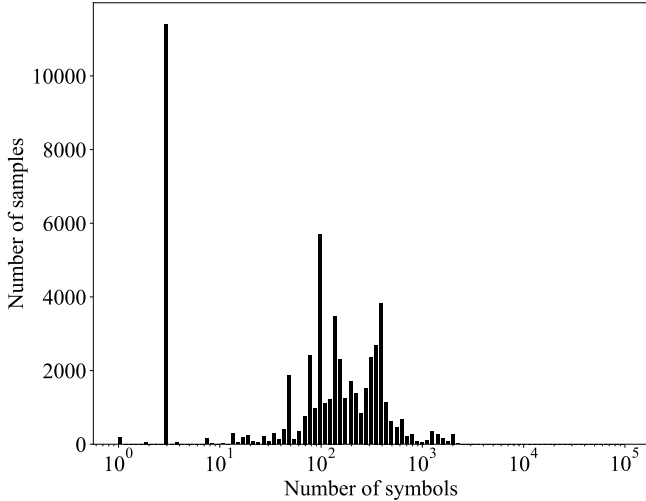


Figure 3: Number of imported symbols.

Specifically, the average number of symbols imported by these binaries is 190.60. The sample with the highest number of imported symbols, totaling 2,895 symbols, belongs to the puagent malware family (identified as a Trojan); in contrast, several samples from different families (including evilquest, amos, aobokeylogger, among others) import zero symbols. Additionally, 85.99% of the samples have their symbols removed, a common technique used by malware developers to reduce binary size and increase analysis complexity. A small fraction of the binaries 15 samples in total required manual inspection because radare2 parsed them incompletely or incorrectly. Files identified during this manual review were incorporated into the final dataset. Figure 3 shows the distribution of imported symbols per sample.

Additionally, we extracted the underlying programming languages of these binaries with radare2. The most common languages identified were C (21,173 samples), Objective-C (16,716), Objective-C with blocks (10,522), C++ (7,575), Swift (4,462), and Go (1,037). Blocks, introduced in 2008 with macOS 10.5, extend C, C++, and Objective-C with closure semantics. They allow code segments to be treated as first-class values, allowing the capture of variables from the surrounding scope and storing them in collections such as NSArray or NSDictionary. This feature simplifies tasks such as enumerating collections and improves the expressiveness of the programming language (DeveloperApple, 2014).

Binary Size Variability. We also analyzed the variations in binary file sizes within our dataset. The average binary file size is 2MB, with the largest being 256.02MB. This largest file is classified as SINGLETON by AVClass, indicating that there was no consensus among anti-malware engines to assign it to a specific malware family. The smallest binaries are 1.70KB in size; one belongs to the installcore malware family, while the other is also classified as SINGLETON. In general, macOS malware appears to have similar binary sizes to Linux and Windows malware, typically ranging between 100 and 200 KB (Cozzi et al., 2018; Bayer et al., 2009).

Entropy Analysis for Packed Binaries. Entropy analysis is a

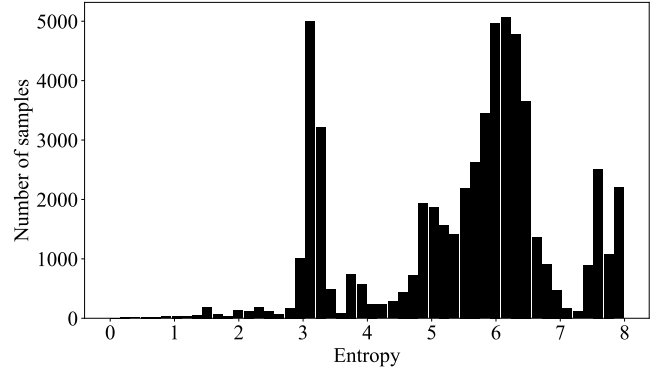


Figure 4: Entropy distribution over the dataset.

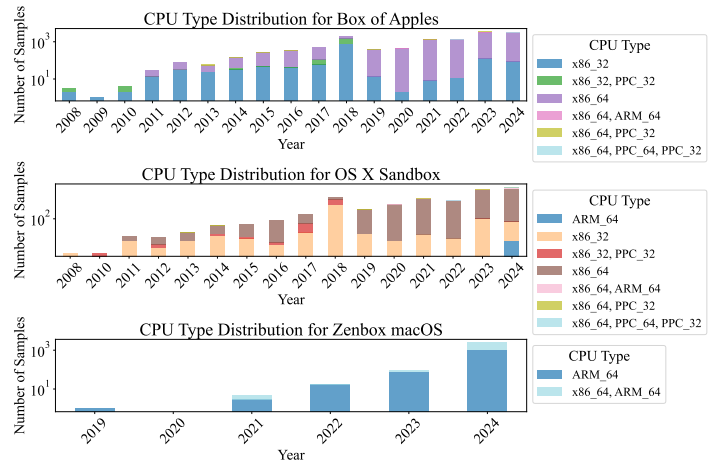


Figure 5: Annual distribution of malware samples by VT sandboxes.

valuable technique for identifying potentially packed binaries or encrypted sections within executables. In our dataset, the majority of samples appear to be uncompressed, with a mean entropy of 5.48. Figure 4 illustrates the overall distribution of entropy across the analyzed samples. However, 18.34% of samples exhibit entropy levels characteristic of packing or encryption.

Notably, none of these high-entropy samples matched the signatures of known packers, as verified using the DIE tool. This suggests the use of custom packing mechanisms, possibly designed specifically for macOS malware. This observation marks a significant difference from patterns observed on other operating systems. For example, previous work on Linux malware indicates that nearly 99% of packed binaries were compressed using UPX or UPX variants (Cozzi et al., 2018). Similarly, research on Windows malware revealed that 75.8% of packed samples used a known packer (Cheng et al., 2018).

Malware Families. As we noted in Section 3.3, dynamic behavioral information is available for 21,967 samples in our dataset. Of these, 15,316 have been labeled by the AVClass tool, resulting in 403 distinct families. Table 3 presents the top 10 families, along with the number of samples in each. We have removed the SINGLETON category from the table as it does not represent an actual family but rather unlabeled data and is also excluded from the distinct family count.

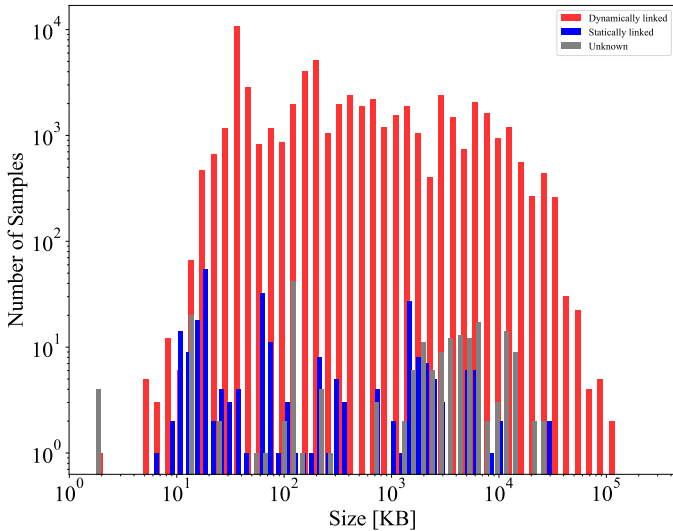


Figure 6: File size distribution of macOS malware in the dataset. *Unknown* files are those that lack the necessary flags in their headers, such as DYLDLINK (dynamically linked Mach-O files) or NOUNDEFS (statically linked Mach-O files without the DYLDLINK flag), or have no flags present whatsoever.

For the remaining 35,669 samples, dynamic behavioral information could not be extracted. However, AVClass successfully labeled 25,320 of these samples, categorizing them into 281 families. Table 3 provides details of the top 10 families, along with sample counts for each, among those lacking dynamic information. As before, we have removed the SINGLETON category.

Linking Status. We analyzed the linking method of the binaries and found that 53,447 samples are dynamically linked, as indicated by the presence of the DYLDLINK flag. In contrast, 257 samples are statically linked (without the DYLDLINK flag, but marked with the NOUNDEFS flag). Another 3,932 samples could not be classified due to the absence of the relevant flags. Figure 6 shows the distribution of file sizes in the dataset, along with the linking status of the binaries.

Statically linked binaries embed all necessary code at compile time, resulting in larger file sizes and preventing code page sharing at runtime. For this reason, Apple recommends using dynamic linking. Although statically linked binaries are relatively uncommon, they can still be generated. Creating them often requires explicit options to the load command and may suggest attempts at obfuscation or other non-standard behavior.

5. Insights into Tactics, Techniques, and Procedures

In this section we present a detailed overview of the malware attack cycle, based on the MITRE ATT&CK framework. The malware attack cycle consists of several phases that a malware sample goes through, from initial access (Ini) to system compromise/execution (Exe), persistence (Per), privilege escalation (Pri), defense evasion (Def), credential access (Cre), discovery (Dis), lateral movement (Lat), collection (Col), command and control (Com), exfiltration (Exf), and ultimately achieving its goal (Imp). Our goal is to highlight the different phases that

Table 3: Top 10 malware families with and without behavior information, as classified by AVClass.

With behavioral data		Without behavioral data	
Family name	Count (%)	Family name	Count (%)
amos	2,953 (13.44%)	bundlore	10,349 (29.01%)
bundlore	1,596 (7.27%)	tirrip	4,188 (11.74%)
tirrip	1,077 (4.9%)	amos	1,913 (5.36%)
cimpli	956 (4.35%)	cimpli	1,536 (4.31%)
flashback	862 (3.92%)	evilquest	1,452 (4.07%)
mackontrol	739 (3.36%)	synataeb	1,343 (3.77%)
genieo	645 (2.94%)	imobie	778 (2.18%)
maxofferdeal	628 (2.86%)	xcsset	759 (2.13%)
lador	475 (2.16%)	lador	739 (2.07%)
installcore	338 (1.54%)	adagent	469 (1.31%)
Total	10,242 (46.62%)	Total	13,526 (65.96%)

macOS malware goes through, detailing what actions the malware performs at each stage and how these actions are executed.

Figure 7 shows the MITRE ATT&CK navigator matrix (MITRE Corporation, 2024), where each technique is represented with a gradient scale reflecting its overall frequency of use across all malware samples analyzed. The samples selected for this matrix are those with the highest number of techniques within each malware family. This matrix provides an overview of the categorization of relevant tactics and techniques within the framework. Additionally, Figure 8 illustrates the timeline of each technique, organized by tactic, from its first documented use to its most recent appearance. This visualization helps track the evolution of each technique, revealing whether certain techniques have been discontinued or whether new techniques are emerging.

The following tables provide a detailed overview of the techniques employed by macOS malware (relative to the total dataset) in various MITRE ATT&CK tactics. Table 4 presents the ten most common techniques observed across the entire dataset, including their association with specific tactics, while Table 5 indicates the percentage of samples that exhibit at least one technique for each tactic and identifies the three most frequently observed techniques within each tactic.

To guide our analysis, and aligned with the malware attack cycle, we propose the following research questions:

- RQ1.-** *What techniques does macOS malware most frequently use during the initial access and execution phases? (see Section 5.1).*
- RQ2.-** *How does macOS malware achieve persistence and privilege escalation? (see Section 5.2).*
- RQ3.-** *What defense evasion techniques does macOS malware most frequently use? (see Section 5.3).*
- RQ4.-** *What methods does macOS malware use to access credentials, discover the environment, and move laterally within a network? (see Section 5.4).*
- RQ5.-** *What files, data, and credentials does macOS malware typically collect? (see Section 5.5).*

Table 4: Top 10 most frequently used TTPs in our dataset.

Technique	MITRE ATT&CK Tactic											Count	
	Ini	Exe	Per	Pri	Def	Cre	Dis	Lat	Col	Com	Exf		Imp
<i>Encrypted Channel</i> (T1573)										✓			14,628
<i>Application Layer Protocol</i> (T1071)										✓			13,913
<i>Non-Application Layer Protocol</i> (T1095)										✓			13,357
<i>System Information Discovery</i> (T1082)							✓						11,712
<i>Security Software Discovery</i> (T1518.001)							✓						7,282
<i>Scripting</i> (T1064)		✓			✓								5,945
<i>Hidden Files and Directories</i> (T1564.001)					✓								5,076
<i>Code Signing</i> (T1553.002)					✓								4,972
<i>Invalid Code Signature</i> (T1036.001)					✓								4,415
<i>Virtualization/Sandbox Evasion</i> (T1497)					✓		✓						4,246

Table 5: Distribution of malware samples and the 3 most prevalent techniques for each MITRE ATT&CK tactic. (*n/a*) indicates absence of tactics or techniques.

Tactic	Count (%)	Technique	Count (%)
Ini	(<i>n/a</i>)	–	–
Exe	6,526 (42.19%)	<i>Scripting</i> (T1064)	5,945 (38.43%)
		<i>Command and Scripting Interpreter</i> (T1059)	3,360 (21.72%)
Per	4,071 (26.32%)	<i>AppleScript</i> (T1059.002)	3,089 (19.97%)
		<i>LC_LOAD_DYLIB Addition</i> (T1546.006)	1,684 (10.89%)
		<i>Launch Agent</i> (T1543.001)	553 (3.57%)
Pri	4,144 (26.79%)	<i>Launch Daemon</i> (T1543.004)	166 (1.07%)
		<i>LC_LOAD_DYLIB Addition</i> (T1546.006)	1,684 (10.89%)
		<i>Launch Agent</i> (T1543.001)	553 (3.57%)
Def	11,857 (76.65%)	<i>Launch Daemon</i> (T1543.004)	166 (1.07%)
		<i>Scripting</i> (T1064)	5,945 (38.43%)
Cre	952 (6.15%)	<i>Hidden Files and Directories</i> (T1564.001)	5,076 (32.81%)
		<i>Code Signing</i> (T1553.002)	4,972 (32.14%)
Dis	12,978 (83.90%)	<i>Input Capture</i> (T1056)	601 (3.89%)
		<i>Keychain</i> (T1555.001)	298 (1.93%)
		<i>Network Sniffing</i> (T1040)	252 (1.63%)
Lat	22 (0.14%)	<i>System Information Discovery</i> (T1082)	11,712 (75.71%)
		<i>Security Software Discovery</i> (T1518.001)	7,282 (47.07%)
		<i>Virtualization/Sandbox Evasion</i> (T1497)	4,246 (27.45%)
Col	613 (3.96%)	<i>Remote Desktop Protocol</i> (T1021.001)	21 (0.14%)
		<i>Remote Services</i> (T1021)	1 (0.01%)
Com	14,967 (96.75%)	(<i>n/a</i>)	–
		<i>Input Capture</i> (T1056)	601 (3.89%)
		<i>Data from Local System</i> (T1005)	12 (0.08%)
Exf	1,407 (9.10%)	<i>Archive Collected Data</i> (T1560)	5 (0.03%)
		<i>Encrypted Channel</i> (T1573)	14,628 (94.56%)
		<i>Application Layer Protocol</i> (T1071)	13,913 (89.94%)
Imp	4 (0.03%)	<i>Non-Application Layer Protocol</i> (T1095)	13,357 (86.35%)
		<i>Exfiltration Over Alternative Protocol</i> (T1048)	1,404 (9.08%)
		<i>Exfiltration Over C2 Channel</i> (T1041)	7 (0.05%)
		(<i>n/a</i>)	–
		<i>Service Stop</i> (T1489)	4 (0.03%)
		(<i>n/a</i>)	–

RQ6.- *How do macOS malware samples exfiltrate collected information, maintain communication with their command and control infrastructure, and what impact do these malicious actions have?* (see Section 5.6).

We provide a detailed answer to each question below.

5.1. RQ1: Initial Access and Execution

The initial access and execution phases involve gaining entry into the target system and subsequently executing malicious

code to establish control and carry out the adversary’s objectives. *Initial Access* typically occurs through methods such as vulnerability exploitation or phishing, which occur prior to malware deployment. No specific initial access techniques were identified in our analysis, as the dataset is primarily focused on malware samples that have already been obtained and are being analyzed for their behavior once executed on a system (i.e., we focus on malware behavior post-compromise). Therefore, our analysis focuses solely on the *Execution* phase.

Table 5 shows that 42.19% of the analyzed samples used at least one technique related to the *Execution* phase. The most commonly used techniques include *Scripting* (T1064)¹, *Command and Scripting Interpreter* (T1059), and *AppleScript* (T1059.002), as highlighted in Table 5. Note that in the overall ranking of techniques, depicted in Table 4, *Scripting* ranks sixth.

Table 6 presents the top ten most frequently executed commands among the analyzed samples that employ this tactic, with `sh` being the most frequent. This command serves as a versatile interpreter, commonly used to execute shell commands and scripts directly within the target system. Additionally, malware often leverages `sh -c` to execute compound commands, which will be explained later. Commands such as `chmod` and `defaults` are also frequently used to modify permissions and seamlessly run scripts.

Another commonly used TTP, *AppleScript*, is a macOS-specific scripting language used for system administration, task automation, and interaction with other applications. These scripts can be executed via the `osascript` command, the third most used command. Notably, `sh -c osascript` was observed in 2,301 (10.48%) of the analyzed samples employing this tactic.

Table 7 presents the libraries loaded in dynamically linked executables, which we specifically analyze because certain li-

¹The *Scripting* technique has been deprecated in the latest MITRE ATT&CK framework and now redirects to the *Command and Scripting Interpreter* technique.

Table 6: Top 10 most common shell commands executed by macOS malware.

Command	Count (%)
sh	5,009 (22.80%)
mkdir	3,191 (14.53%)
osascript	2,931 (13.34%)
dscl	2,595 (11.81%)
/usr/bin/syslog	2,537 (11.55%)
system_profiler	2,378 (10.83%)
sw_vers	1,884 (8.58%)
rm	1,665 (7.58%)
chmod	1,549 (7.05%)
defaults	1,234 (5.62%)

libraries contain execution-related functions. The most commonly used library is `libSystem`, which in turn includes other key libraries: `libc` (which provides functions used by C programmers on all platforms), `libinfo` (NetInfo services), `libkvm` (kernel virtual memory), `libm` (mathematical functions), and `libpthread` (POSIX threads). The importance of `libpthread` is further emphasized by the symbols `pthread_mutex_lock` and `pthread_mutex_unlock`, which rank tenth among the most commonly used symbols, as shown in Table 9 allowing the execution of concurrent tasks. Other symbols, such as `fopen`, `fclose`, and `fwrite` are also crucial during the execution phase, particularly when malware dynamically writes or modifies files before executing them.

The second most commonly used library is `libobjc`, which supports the dynamic properties of the Objective-C language. Symbols such as `objc_msgSend` (44.25%) are frequently used to invoke Objective-C methods, reflecting the language’s messaging-based model. While `libobjc` is specific to Objective-C, these runtime mechanisms may also appear in mixed-language applications or when C code interacts with Objective-C APIs. These results demonstrate that Objective-C remains one of the top language for writing Mach-O malware, given its important role in developing macOS applications.

The remaining libraries are primarily associated with C++ and Swift. C++ libraries such as `libc++` (which ranks third). Although `libgcc_s` is also present, it is a GCC runtime library used for low-level operations like exception handling. Regarding Swift, although it has largely replaced Objective-C as Apple’s language of choice in benign software, its presence in malware remains minimal, although it has begun to emerge slightly in the analyzed samples. Swift libraries such as `libswiftCoreFoundation`, `libswiftAppKit`, `libswiftCore`, `libswiftFoundation`, `libswiftDispatch`, and `libswiftObjectiveC` allow malware to take advantage of core Swift features, manage system resources, and seamlessly interact with Objective-C. Other Swift libraries such as `libswiftDarwin`, `libswiftIOKit`, and `libswiftXPC` provide support for system-level operations and secure inter-process communication, while `libswiftCoreGraphics`, `libswiftMetal`, and `libswiftQuartzCore` are used to build GUIs on macOS.

Table 7: Top 20 libraries included by dynamically linked executables in macOS malware.

Library	Count (%)
<code>libSystem.B</code>	55,578 (96.43%)
<code>libobjc</code>	27,880 (49.37%)
<code>libc++</code>	20,470 (35.52%)
<code>libz</code>	16,980 (29.46%)
<code>libsqlite3</code>	8,480 (14.71%)
<code>libswiftCore</code>	3,238 (5.62%)
<code>libswiftFoundation</code>	3,230 (5.60%)
<code>libswiftDispatch</code>	3,155 (5.47%)
<code>libswiftObjectiveC</code>	3,117 (5.41%)
<code>libresolv</code>	2,710 (4.70%)
<code>libgcc_s</code>	2,672 (4.64%)
<code>libswiftCoreGraphics</code>	2,606 (4.52%)
<code>libswiftDarwin</code>	2,583 (4.48%)
<code>libswiftCoreFoundation</code>	2,536 (4.40%)
<code>libswiftIOKit</code>	2,515 (4.36%)
<code>libswiftXPC</code>	1,975 (3.42%)
<code>libswiftAppKit</code>	1,949 (3.38%)
<code>libswiftQuartzCore</code>	1,846 (3.20%)
<code>libswiftCoreImage</code>	1,826 (3.17%)
<code>libswiftCoreData</code>	1,826 (3.17%)

In addition to these standard system libraries, macOS malware samples also often import high-level system frameworks to facilitate interaction with various operating system components. As shown in Table 8, frameworks such as `CoreFoundation`, `Foundation`, and `IOKit` are frequently used in the dataset. This indicates that many malware samples rely on native macOS APIs not only for process execution and control, but also for interacting with hardware components and performing user-space operations.

Takeaways from RQ1

Our analysis shows that 42.19% of the analyzed macOS malware samples use at least one execution-related technique, with scripting (T1064), command interpreter (T1059), and *Apple scripting* (T1059.002) being the most common. Malware often employs commands such as `sh`, `chmod`, and `osascript` to run scripts and modify permissions, highlighting the importance of script-based execution in macOS attacks. Objective-C remains the dominant language for macOS malware, leveraging libraries such as `libobjc` and symbols like `objc_msgSend` for efficient execution and interaction with core system functionalities, while Swift has only just begun to emerge. Symbols such as `fopen`, `fclose`, and `fwrite` are frequently used to dynamically write or modify files, indicating that file manipulation is a key part of the malware execution strategy.

Table 8: Top 20 Apple higher-level frameworks included by dynamically linked executables in macOS malware.

Framework	Count (%)
CoreFoundation	34,689 (60.19%)
Foundation	27,951 (48.50%)
IOKit	23,246 (40.33%)
libsysteurity	20,506 (35.58%)
CoreServices	17,398 (30.19%)
AppKit	14,301 (24.81%)
ApplicationServices	14,289 (24.79%)
CFNetwork	12,643 (21.94%)
CoreGraphics	8,133 (14.11%)
Cocoa	7,326 (12.71%)
SystemConfiguration	7,292 (12.65%)
Carbon	6,524 (11.32%)
QuartzCore	2,191 (3.80%)
WebKit	2,043 (3.54%)
ImageIO	1,494 (2.59%)
SafariServices	1,398 (2.43%)
DiskArbitration	1,368 (2.37%)
CoreText	1,263 (2.19%)
ServiceManagement	933 (1.62%)
HIServices	779 (1.35%)

5.2. RQ2: Persistence and Privilege Escalation

The malware establishes persistence on a target system through several techniques, often aiming to escalate privileges for greater control. *Persistence* and *privilege escalation* tactics were identified in 26.32% and 26.79% of the analyzed samples, respectively, as shown in Table 5. The top techniques for both phases are *LC_LOAD_DYLIB Addition* (T1546.006), *Launch Agent* (T1543.001) and *Launch Daemon* (T1543.004), as shown in Table 5. The *LC_LOAD_DYLIB* header in Mach-O binaries specifies which dynamic libraries (dylibs) to load, allowing adversaries to modify the headers to execute malicious dylibs persistently. Launch agents are executed once a user logs in, allowing malware to interact with the user’s session, often through entries in */Library/LaunchAgents* or */Library/LaunchAgents*.

Commands such as *chmod* and *defaults* (ranked ninth and tenth in Table 6, respectively) modify permissions, while *dscl* (ranked fourth) manages Directory Services, which serve as a central repository for storing users and passwords, particularly in enterprise environments. Similarly, commands such as *mkdir* (or *sh -c mkdir*, used by 1.07% of samples) and *rm* (ranked second and eighth, respectively), are used to create directories or delete detection artifacts, aiding malware persistence.

The *libsqlite3* library (ranked fifth in Table 7) can be used by malware to access and modify local databases, facilitating persistent access to the system. Additionally, symbols such as *fopen* (41.29%), *fclose* (42.22%), and *fwrite* (40.79%) are also used for persistence by creating or modifying system files, such as the launch agents mentioned earlier. In addition, high-

level frameworks such as *ServiceManagement*, *IOKit*, and *SystemConfiguration* (listed in Table 8) are also frequently imported. These frameworks allow malware to manage startup daemons, interact with hardware components, and configure system services, all of which are commonly used for establishing and maintaining persistence.

Table 9: Top 10 most common symbols imported by dynamically linked executables.

Symbol	Count (%)
<i>dyld_stub_binder</i>	47,563 (82.52%)
<i>malloc</i>	36,613 (63.52%)
<i>free</i>	34,201 (59.34%)
<i>memcpy</i>	33,613 (58.32%)
<i>strlen</i>	32,904 (57.09%)
<i>exit</i>	31,560 (54.76%)
<i>pthread_mutex_lock</i>	29,740 (51.60%)
<i>pthread_mutex_unlock</i>	29,740 (51.60%)
<i>strcmp</i>	27,998 (48.58%)
<i>pthread_create</i>	25,214 (43.75%)

From the list of open files (collected from VT reports), we specifically selected those in the */etc* directory, as these files often contain essential configuration data, such as system settings and user credentials, which are frequently targeted by malware to establish persistence or escalate privileges. The top 10 opened */etc* files (see Table 10) reveal files such as */etc/master.passwd*, */private/etc/passwd*, */etc/profile*, and */etc/bashrc* that are often used to gain elevated permissions or ensure malware runs during startup. Files such as */private/etc/ssl/openssl.cnf* can be used to intercept authentication or encrypted data, facilitating privilege escalation.

Takeaways from RQ2

Persistence and privilege escalation are observed in approximately 26% of the analyzed macOS malware samples. The most common techniques include addition of dynamic libraries (T1546.006) and abuse of launcher

Table 10: Top 10 most common */etc* system files opened by macOS malware.

File	Count (%)
<i>/etc/master.passwd</i>	13,930 (63.41%)
<i>/etc/motd</i>	2,495 (11.36%)
<i>/etc/profile</i>	2,494 (11.35%)
<i>/etc/bashrc</i>	2,494 (11.35%)
<i>/etc/bashrc_Apple_Terminal</i>	2,494 (11.35%)
<i>/private/etc/passwd</i>	2,272 (10.34%)
<i>/etc/services</i>	1,991 (9.06%)
<i>/private/etc/services</i>	1,811 (8.24%)
<i>/private/etc/group</i>	1,229 (5.59%)
<i>/private/etc/ssl/openssl.cnf</i>	1,058 (4.82%)

agents (T1543.001), which allow malware to load malicious dynamic libraries or execute tasks upon user login, respectively. Commands like `chmod`, `defaults`, `dscl`, `mkdir`, and `rm` are frequently used to modify permissions, manage directory services, and create or remove files, supporting persistence. The `libsqlite3` library and file manipulation symbols such as `fopen`, `fclose`, and `fwrite` are employed to modify system files or access local databases, enhancing persistent access. Additionally, malware target configuration files in the `/etc` directory to modify system settings and user credentials, ensuring elevated permissions and persistence through system reboots.

5.3. RQ3: Defense Evasion

Defense evasion involves several techniques that adversaries use to avoid detection. Figure 8 illustrates that defensive evasion tactics encompass the widest range of techniques, with an increasing diversity of methods adopted over time. This tactic ranks third among all MITRE ATT&CK tactics (see Table 5), which may partly explain why behavioral data was available for only 38.11% of the samples analyzed. As presented in Table 5, the most prevalent techniques during this phase are *Scripting* (T1064), *Hidden Files and Directories* (T1564.001), and *Code signing* (T1553.002).

Scripting remains a popular method to avoid detection, as it can be executed directly in the macOS shell. This allows malware to identify whether it is in a sandbox or whether analysis tools are active, and subsequently kill all processes or self-destruct to leave no trace. Commands such as `kill` (run by 5.04% of the samples), `sh -c kill` (used by 178 samples, or 0.81%), `ps` (used by 1.88% of samples), and `rm` (used by 7.58% of samples) aid in the evasion process. Some commands are present in Table 6. Additionally, 32.81% of samples employed the *Hidden Files and Directories* technique (T1564.001), which hides files or folders by prefixing their names with a dot (‘.’) character, making them invisible by default in Finder (macOS file explorer) and standard command-line utilities such as `ls`. Files can also be marked with the `UF_HIDDEN` flag, which hides them from Finder while still allowing visibility in the terminal.

In addition to the techniques described above, macOS malware frequently use *Code signing* (T1553.002) as a defense evasion mechanism. By embedding a signature, or at least mimicking the expected structural components of a valid signature, malicious binaries can bypass superficial integrity checks that only verify the presence of a signature, rather than its cryptographic validity.

On macOS, every signed Mach-O binary includes a `LC_CODE_SIGNATURE` load command, which points to a data structure known as a *superblob* (a container format with the magic value `0xFADE0CC0`) located at the end of the binary (Levin, 2016; HackTricks, 2024). This superblob aggregates several constituent blobs, the most common being: (i) *CodeDirectory*, a manifest containing per-page (typically 4 KB) hashes and hashes of special metadata slots

(e.g., slot `-1` for `Info.plist`, `-5` for `Entitlements`, `-2` for `Requirements`, and `-3` the `CodeResources` XML file); (ii) *Entitlements*, XML- and DER-encoded blobs that define the execution permissions granted to the binary; (iii) *Requirements*, a set of compiled rule set that dictate the conditions under which the code signature is considered valid; and (iv) *CodeResources*, an XML file (`_CodeSignature/CodeResources`) containing hashes of all resource files within application package. These structures can be forged or manipulated by adversaries to emulate legitimate binaries, thus evading static and dynamic defenses that rely on code signing as a sign of trust.

Table 11 shows that only 0.84% of the analyzed samples have a valid code signature, while an additional 0.22% include an invalid certificate. Malware with invalid certificates can evade basic detection systems that do not apply strict cryptographic validation. These evasion techniques typically involve copying signature metadata from legitimate applications and reusing it in unsigned or modified malware. Although these files fail digital signature verification, they may appear legitimate to end users or be mishandled by security tools that rely on superficial signature checks.

Furthermore, 1.40% of samples lacked embedded metadata, such as the `Info.plist` file (categorized as “missing plist” in Table 11), a common component in properly signed macOS applications. The absence of this metadata can contribute to bypassing basic verification routines, especially on systems that rely on the presence of expected file structures rather than their integrity. For instance, the `ShLayer` operates without an `Info.plist` file and can execute without triggering quarantine, code signing, or notarization checks. This vulnerability was later addressed by Apple in macOS 11.3 (Jamf, 2021).

In this sense, malware samples with invalid or nonexistent code signatures can exploit vulnerabilities in security products that check only for the presence of the `LC_CODE_SIGNATURE` section or superficially look for a `Developer ID` string. These techniques reinforce the importance of thorough cryptographic validation in signature-based defenses.

Table 11: Breakdown of code signing status among analyzed malware samples.

Code signing status	Count (%)
<i>Not signed</i>	56,204 (97.52%)
<i>Missing plist file</i>	806 (1.40%)
<i>Valid signature</i>	482 (0.84%)
<i>Invalid signature</i>	125 (0.22%)
<i>Untrusted signature</i>	17 (0.03%)
<i>Certificate out of its validity period</i>	2 (0.00%)
Total	57,636 (100.0%)

Virtualization/Sandbox Evasion (T1497) is another common evasion technique, ranking 10th in Table 4, with 27.44% of the analyzed samples employing it. Malware can detect specific virtual machine environments or artifacts and then alter their main behavior or prevent downloading additional payloads. Sleep timers are often used to evade sandbox analysis, with symbols such as `usleep` (imported by 16.19% of the sam-

ples) and `sleep` (12.16%) serving for this purpose. Additionally, the `exit` symbol was imported by 31,560 (54.76%) samples, likely indicating early process termination to evade detection by preventing malicious behavior from being observed.

Another common defense evasion technique involves modifying binaries by packing or stripping them. Although VirusTotal sandboxes do not provide details about packed files, our static analysis reveals these evasion methods (discussed above in Section 4). Based on the file entropy, approximately 18.35% of the binaries were packed, making static analysis more difficult. This aligns with the *Obfuscated Files or Information: Software Packing* technique (T1027.002). Furthermore, 85.99% of the binaries were stripped, meaning that debugging symbols were removed to hide valuable information about function names and variable identifiers. This practice aligns with the *Obfuscated Files or Information: Stripped Payloads* technique (T1027.008).

The `libz` library (ranked fourth in Table 7) is also used to evade defenses by compressing and decompressing data, thereby altering its structure and reducing its size to evade detection. Additionally, the `unzip` command was also executed by 168 samples, further indicating the use of compression as part of an evasion strategy.

Another important evasion technique involves accessing the `/etc/hosts` file (805 samples). While the file was modified 3 times, accessing `/etc/hosts` suggests the possibility of redirecting network traffic away from monitoring or security servers, thereby attempting to bypass network-based defenses.

Takeaways from RQ3

Defense evasion techniques are imperative to avoid detection, allowing malware to operate undetected for longer periods of time. Popular methods include scripting in the macOS shell (T1064), hiding files (T1562.001), and leveraging or manipulating code signing (T1553.002) to appear legitimate. Virtualization and sandbox evasion (T1497), detected in a significant portion of samples, allow malware to effectively avoid analysis by detecting virtualized environments. Additionally, modifying key system files (such as `/etc/hosts`) further complicate detection efforts. Together, these techniques contribute to defense evasion being one of the most prevalent tactics, with a wide variety of methods employed to overcome security defenses.

5.4. RQ4: Credential Access, Discovery, and Lateral Movement

Adversaries use credential access tactics to obtain sensitive authentication information such as passwords, private keys, and account names. The goal is to gain unauthorized access to privileged information or systems, enabling privilege escalation or lateral movement.

The two most common techniques for credential access observed in our dataset are *Keychain* (T1555.001; observed in 298 samples) and *Input Capture* (T1056; observed in 601 samples).

The Keychain is the macOS credential management system responsible for securely storing account names, passwords, private keys, certificates, and other sensitive information. There are three types of Keychains: the Login Keychain, the System Keychain, and Local Items. Keychains can be accessed and edited using the Keychain Access application or the `security` command-line utility, which was run by 254 samples. By accessing the Keychain, adversaries seek to retrieve credentials that can be reused to gain unauthorized access to additional systems or services. The *Input Capture* technique overlaps with data collection methods (explained below) but is specifically used to capture credentials such as usernames and passwords, typically through keylogging or other methods to intercept input.

Despite the relatively low incidence of credential access techniques, being observed in only 6.15% of the analyzed samples (see Table 5), these methods are critical in an adversary’s lifecycle. Obtaining valid credentials significantly facilitates further lateral movement and privilege escalation.

Adversaries employ discovery techniques to gather information about the target system and internal network. These techniques allow adversaries to better understand their operational environment before taking further action, helping to identify opportunities for lateral movement across the target network while minimizing detection risks. The most common discovery techniques identified in our analysis are presented in Table 5, with the top three being *System Information Discovery* (T1082), *Security Software Discovery* (T1518.001), and *Virtualization/Sandbox Evasion* (T1497).

System Information Discovery (T1082), used by 75.71% of the analyzed samples, is used to gather detailed information about the target’s operating system and hardware, including version, installed patches, service packs, and architecture. This information helps tailor the malware’s subsequent actions. Tools such as `systemsetup`, when run with privileged access, can provide a detailed breakdown of system configuration. *Security Software Discovery* (T1518.001), observed in 47.07% of the samples, allows adversaries to identify installed security software, defensive tools, and sensors. This information helps attackers develop custom lateral movement techniques. While *Virtualization/Sandbox Evasion* (T1497) falls primarily under defense evasion, it also plays an important role in discovery. This technique, used by 27.45% of analyzed samples, helps determine whether the malware is running in a virtualized environment or if there are any analysis artifacts present. This knowledge allows the malware to adapt its behavior accordingly, which could mean preventing further execution or altering payload deployment strategies.

In addition to these techniques, other commands and tools were frequently used for discovery. Commands such as `dsccl` (11.81%; ranked 4th in Table 6) help gather information about user accounts, while `/usr/bin/syslog` (11.55%; ranked 5th) is often used to trace outputs from security software or check for signs of a virtual environment. Similarly, the `system_profiler` command (10.83%; ranked 6th) is used to gather detailed information about system hardware and software. Other commands, such as `sw_vers` (8.58%; ranked 7th),

ioreg (3.71%), and stat (3.88%), further help gather data about system configuration, hardware, and file systems.

Several libraries shown in Table 7 can also help gather information. For instance, `libsqlite3` is used for local data storage, giving attackers the ability to query local databases. Other libraries, such as `libswiftIOKit`, interact with hardware. Similarly, `libswiftXPC` and `libswiftCoreFoundation` provide information about running processes and system configurations.

Table 10 shows some of the most frequently accessed files that are important in the discovery process. Files such as `/etc/master.passwd` and `/private/etc/passwd` list user accounts and associated attributes, providing attackers with valuable information about system users. Additional files such as `/etc/group`, `/etc/services`, `/etc/hosts`, and `/etc/profile` contain configuration and environment information, helping attackers understand privilege levels, network services, and other system settings.

Despite their importance in the adversary’s lifecycle, no lateral movement techniques were identified in our analysis. This is likely because our dataset focuses on binaries and malware analysis reports from samples executed in isolated environments, which lack the necessary conditions to observe lateral movement. Lateral movement typically involves the propagation from an initially compromised system to other devices within a network, requiring an active and interconnected network environment. Given our focus on static and dynamic analysis of malware samples under controlled conditions, opportunities to observe these techniques were limited, resulting in an absence of lateral movement behavior in our findings.

Takeaways from RQ4

Credential access, discovery, and lateral movement are interconnected phases that are critical to an adversary’s operational success. Credential access techniques were observed in a small proportion of the analyzed samples (6.15%), but play an essential role in obtaining sensitive information such as passwords and private keys, facilitating unauthorized access to systems. Recovering such credentials can significantly improve adversaries’ capabilities for privilege escalation and lateral movement. The discovery tactics analyzed reveal the importance of gaining detailed knowledge of the target system and network environment. Common techniques include gathering system information (T1082), identifying installed security software (T1518.001), and determining whether malware is running in a virtualized environment (T1497). These methods help adversaries assess the target environment and identify potential opportunities for further compromise. However, no lateral movement techniques were detected, likely due to the controlled environment in which the malware samples were analyzed, which lacked the necessary conditions for lateral movement across networked systems.

5.5. RQ5: Collection

The *Collection* tactic involves techniques used to gather information from various relevant sources to advance an adversary’s goals. These sources can include different types of drives, browsers, email, and even audio or video data. Common collection methods include screen capture, recording keyboard input, and searching for specific files. Once the information has been gathered, the next logical step is the exfiltration phase, which involves stealing data from the compromised network (discussed in the next section). Finally, credential access techniques are employed to obtain sensitive authentication information such as usernames, passwords, and other credentials (detailed in Section 5.4).

The samples in our dataset primarily employed two techniques (as shown in Table 5). The most common technique was *Input Capture* (T1056), observed in 601 samples, where user input such as keystrokes is captured to obtain credentials or gather other useful information. Users often provide credentials during regular interactions with login pages or system prompts, making this method effective for credential harvesting. The second observed technique was *Data from Local System* (T1005), present in 12 samples, which involves searching local system sources such as file systems and local databases for sensitive data before exfiltration. Tools such as `find` and `grep` have been employed in 1,131 and 778 samples, respectively, to search for specific files or text strings. Additionally, some samples used the *Archive Collected Data* (T1560) technique to compress or encrypt collected data before exfiltration; this was observed in samples that often relied on the `libz` library (observed in 16,980 samples) or `tar` command (observed in 334 samples). Furthermore, system frameworks such as `ImageIO`, `SafariServices`, and `WebKit`, as shown in Table 8, allow malware to collect visual data, retrieve browser-related information, and potentially capture web-based credentials. Finally, only one sample was found using the *Screen Capture* technique (T1113), emphasizing its limited use within the known macOS threat landscape.

Takeaways from RQ5

The most frequently observed techniques in our dataset involved capturing user input (T1056) and searching local system sources (T1005). The first technique, which was seen in 601 samples, is primarily used for credential harvesting, as users often enter sensitive information during interactions with login pages or system messages. The second technique, seen in 12 samples, involves searching local sources such as file systems and databases to obtain sensitive data before exfiltration. Tools such as `find` and `grep` were employed in 1,131 and 778 samples, respectively, for this purpose. Additionally, several samples were found to compressing or encrypting the collected data (T1560), relying on tools such as the `tar` command (334 samples) and the `libz` library (18,212 samples). Interestingly, capturing data from screen (T1113) was only observed in a single

sample, suggesting its limited use in the current macOS threat landscape.

5.6. RQ6: Exfiltration, Command and Control, and Impact

After the information gathering phase, the next step in an adversary’s strategy is typically the *exfiltration* phase, during which techniques are used to steal data from the compromised network. In our dataset, *Exfiltration Over Alternative Protocol* (T1048) was the most frequently observed technique, present in 1,404 samples. This method involves transferring data over protocols other than the main command and control (C2) channel, such as FTP, SMTP, HTTP/S, DNS, and SMB. For instance, the `curl` command, which can upload files over multiple protocols, was executed in 1,042 samples (4.76%). Similarly, `rsync` was used in 1,177 samples to send data directly to an external server. On the other hand, *Exfiltration Over C2 Channel* (T1041) was found in only 7 samples, indicating a less frequent use of this method, where data is exfiltrated over the same channel as the C2 communication. These findings demonstrate how adversaries prioritize alternative communication channels to exfiltrate data without triggering alarms in their C2 infrastructure.

This reliance on alternative communication methods leads to the central role of *Command and Control* tactic, the most frequent in all samples analyzed, with almost all samples (96.75%) using C2-related techniques (see Table 5). This is quite expected, as C2-related techniques allow for persistent and covert communication between infected devices and remote servers. This tactic allows the malware to adapt its behavior based on real-time information it receives from the C2 infrastructure, ensuring malicious business continuity. Notably, the three most frequently observed TTPs fall into this tactic, as shown in Table 4. These techniques are *Encrypted Channel* (T1573), used in 94.56% of samples; *Application Layer Protocol* (T1071), observed in 89.94%; and *Non-Application Layer Protocol* (T1095), used in 86.35% of samples.

The *Encrypted Channel* technique is widely used by malware to hide C2 traffic by employing encryption rather than relying on inherent protocol protections. Relevant symbols supporting this technique include `SSL_read` (imported by 553 samples) and `SSL_write` (imported by 554 samples), which facilitate encrypted communication over SSL/TLS channels. Both symbols belong to the `libssl` library (loaded by 491 samples). Commands such as `curl` (run in 1,042 samples) and `openssl` (run in 249 samples) leverage these encryption techniques for secure communication with the C2. Another third-party library that provides essential encryption and decryption capabilities is `libcrypto` (loaded by 495 samples).

The *Application Layer Protocol* technique involves combining malicious traffic with legitimate network traffic by using standard OSI application layer protocols. The `libcurl` library, which is loaded by 913 samples, facilitates data transfer over multiple protocols, making it a common tool for malware that leverages this technique. In contrast, the *Non-Application Layer Protocol* technique uses lower OSI layer protocols for

communication with the C2, which can help evade detection by blending in with less obvious traffic patterns.

To better understand the external connections established by the samples, we analyzed all available network-related information. Both IP addresses and domains were checked using VirusTotal and the Neutrino API (see §3.3). In total, 4,966 domains were analyzed: VirusTotal flagged 203 as malicious, while the Neutrino API identified 265. All domains were compared with DGArchive; however, no matches were found with known algorithmically generated domains.

Regarding IP addresses, we examined 18,922 unique IP addresses, of which VirusTotal flagged 142 as malicious. The Neutrino API provided a more detailed classification of suspicious activity, summarized in Table 12. This table lists the 653 IP addresses identified in the blocklists and their corresponding classifications: *VPN* indicates the IP belongs to a public VPN provider; *Exploit bot* refers to an IP address that hosts an exploit-hunting bot or runs exploit-scanning software; *proxy* refers to an anonymous web or HTTP proxy; *Bot* denotes an IP address that hosts a malicious bot or is part of a general botnet; *spam bot* refers to an IP address associated with a spam bot; *Tor* indicates an IP address that is a Tor node or runs a Tor-related service; *spider* designates a hostile web spider or crawler; *hijacked* refers to an IP address that is part of a network block hijacked or controlled by a criminal organization; *dshield* highlights IP addresses flagged as significant attack sources by DShield, a community-based collaborative firewall log correlation system; *malware* refers to IP addresses that distribute or execute malware (other than spyware); and *spyware* refers to IP addresses involved in distributing or executing spyware.

Table 12: Classification of IP addresses identified by Neutrino API.

IP type	Count	IP type	Count
<i>VPN</i>	380	<i>Spider</i>	12
<i>Exploit bot</i>	202	<i>Hijacked</i>	11
<i>Proxy</i>	196	<i>Dshield</i>	8
<i>Bot</i>	128	<i>Malware</i>	5
<i>Spam bot</i>	150	<i>Spyware</i>	3
<i>Tor</i>	52		

A total of 9,428 samples issued HTTP GET requests, while 18,189 connected over port 443, indicating a heavy reliance on HTTPS. The most frequent Content Types were `application/ocsp-response` (5,970 samples) and `text/html` (3,459 samples), both indicative of web interactions potentially tied to command and control infrastructure. Additionally, 1,922 samples requested content with the HTTP Content Type set to `application/x-apple-plist`, a common format for macOS configuration files, while `application/x-apple-diskimage` appeared in 108 samples. These findings suggest that the malware is specifically designed for Apple environments.

The *Impact* tactic refers to actions taken by adversaries to disrupt the availability, integrity, or confidentiality of systems and data. In our dataset, only two techniques were observed

under this tactic: *Service Stop* (T1489), which was used in four samples to stop critical system services and cause operational disruptions, and *Data Encrypted for Impact* (T1486), which was found in two samples, typically linked to ransomware attacks aimed at making data permanently inaccessible.

Takeaways from RQ6

We found that the most commonly used method for data theft involved data exfiltration through alternative communication channels (T1048), observed in 1,404 samples. This approach allows attackers to transfer data using protocols such as FTP, HTTP/S, or DNS, bypassing the primary communication channel with the control server and reducing the risk of detection. In contrast, data exfiltration through the same channel as command and control communication (T1041) was much less frequent, occurring in only 7 samples. The prevalence of C2 communication, present in almost all samples 96.70%, underlines its essential role in malware operations. Techniques such as encrypted communication (T1573) and mixing malicious traffic with legitimate traffic (T1071 and T1095) enable persistent, covert communication, allowing attackers to maintain control over compromised systems while evading detection. Impact techniques, while less frequently observed, demonstrate adversaries' intentions to disrupt or compromise system integrity. While fewer samples directly targeted system functionality, their actions reflect the potential for severe operational disruption (T1489) or data denial (T1486), often in line with ransomware goals.

6. Platform-Specific Malware Trends and Techniques

Malware targeting different OS exhibits notable differences in behavior and structure. To contextualize our findings on macOS malware, we compared them with information from three benchmark studies analyzing malware for the Linux and Windows platforms (Cozzi et al., 2018; Ugarte-Pedrero et al., 2019; Botacin et al., 2021a).

Architectural Diversity. Linux malware shows greater architectural diversity, with binaries compiled for a wider range of platforms. In contrast, malware targeting macOS and Windows predominantly focuses on Intel architectures, although the use of ARM, particularly Apple Silicon, has grown in the macOS ecosystem in recent years. Mach-O binaries are generally distinguishable by their target OS (macOS versus iOS), while identifying the target OS from ELF binaries is more difficult due to the greater support for UNIX-like systems.

Entropy Analysis and Packaging. Average entropy values also differ across platforms. Linux samples have a higher average entropy (~ 6.0) compared to macOS samples (5.48). In the Windows dataset, 51% of binaries have entropy values above 7, while only 12.17% of macOS binaries exceed this threshold. This discrepancy can be partly attributed to limited support for

packers such as UPX on macOS, which does not natively support the Mach-O format.

Linking Practices. Static linking is significantly more prevalent in Linux malware (80%) than in macOS samples (2.24%). This stark contrast reflects platform-specific design decisions: macOS does not support static linking by default, as explained in Section 4, while static linking is more common and is natively supported on Linux.

Family Diversity and Code Signing. According to AVClass classification, 1,057 unique malware families were identified in the Windows dataset, compared to 492 families in the macOS dataset, suggesting lower family diversity in the latter. Furthermore, AVClass flagged 67.63% of macOS samples as potentially unwanted programs (primarily adware), while only 22% of Windows samples had a similar classification. Code signing was also more prevalent in Windows malware, with 18.3% of samples being signed, compared to only 2.48% on macOS.

Behavioral Characteristics. In terms of behavior, Linux and macOS malware tend to emphasize persistence-related mechanisms, while Windows malware focuses more on data exfiltration and payload retrieval. For macOS samples, only 9.10% exhibit data exfiltration capabilities, and 3.96% perform explicit information collection. In contrast, a staggering 96.75% of macOS samples establish C2 communication, underscoring the importance of remote access and command execution in the macOS threat model. Furthermore, file operations were detected in 38.11% of macOS samples and only 17% of Windows samples, highlighting the tendency of macOS malware to interact with the local file system more frequently.

7. Limitations

Despite the contributions of this work, we acknowledge several limitations to provide a balanced perspective and highlight areas for potential improvement.

Dynamic Analysis with VT Sandbox Reports. The sandbox reports provided by VT are generated using controlled environments that may not fully represent real-world conditions. Limitations here include a lack of transparency into how sandbox environments are set up and the specific detection mechanisms they employ. This lack of information can introduce a degree of uncertainty into understanding how effective or realistic the detection of certain malware behaviors is. Mapping observed behaviors to the ATT&CK framework can be inconsistent, as there is no clear documentation of how VT identifies specific techniques. This can lead to potential misinterpretations or gaps in mapping malware behaviors to MITRE ATT&CK TTPs.

Dataset Limitations. The dataset primarily consists of samples from the post-2023 period, which introduces potential bias into our analysis. As shown in Figure 2, the number of older samples from before 2023 is significantly lower (22.6%), making it difficult to perform a comprehensive historical analysis. This bias could result in an overrepresentation of more recent malware strains and techniques, while older, potentially still

relevant, techniques may be underrepresented. The scarcity of older samples also restricts our ability to track the evolution of macOS malware over time, which could have provided additional insights into long-term trends in malware development and adversarial techniques.

Malware Coverage and TTP Representation. While our dataset is extensive, it may not cover the entire spectrum of macOS malware in circulation. The samples we collected were obtained from publicly available platforms (i.e., VT, VirusShare, and MalwareBazaar), which may have inherent collection biases. These platforms often rely on community contributions and may lack representation of certain malware families or targeted attacks, especially those developed for specific high-value targets. Furthermore, only 26.83% of the dataset presents TTPs. For the remaining 73.17%, TTPs were not found in the VT reports, limiting our ability to fully map and analyze the behavior of these samples.

8. Related Work

The cybersecurity community has traditionally focused on malware targeting Microsoft Windows, resulting in a vast amount of research in that area (see, for instance, recent surveys on this topic (Afianian et al., 2019; Moussaileb et al., 2021; Ling et al., 2023)). Most existing studies focus on specific types of malware or highlight the differences between various malware families. In contrast, relatively few works have focused on analyzing macOS malware, and even fewer have focused specifically on Mach-O files, despite their importance in compromising the macOS environment.

One of the closest studies to our work is (Oosthoek and Doerr, 2019). This study focuses primarily on the MITRE ATT&CK framework, employing API hooking to determine if specific techniques are being used. However, this work was performed on Windows 7, and while it included 951 unique families with one sample per family, this approach may not be fully representative: even within the same family, malware samples can exhibit significantly different behaviors and performance characteristics. For example, there are numerous ways for malware to detect whether it is running in an analysis environment (Afianian et al., 2019; Galloro et al., 2022; Filho et al., 2022), and these variations can affect the results of the analysis. Moreover, another important work is the longitudinal study (Botacin et al., 2021a), which analyzes Windows malware infecting Brazilian banking users from 2012 to 2020. This study highlights rapid evolution in obfuscation, the use of diverse file formats beyond typical PE binaries, and reliance on native system resources, emphasizing the significance of regional and population specific malware campaigns. Lastly, the study (Ugarte-Pedrero et al., 2019) provides an in depth analysis of hundreds of thousands of Windows executables collected in a single day.

Another relevant work that analyzed non-Windows environments is (Cozzi et al., 2018), which focuses on Linux malware. This work analyzed 10,548 samples using static and dynamic techniques, covering aspects such as architectures, headers, commands, and libraries. While similar in scope, our work

differs by leveraging the MITRE ATT&CK and incorporating behavioral data extracted from VT sandbox reports. Additionally, the work in (Chierzi, Veronica and Mercês, Fernando, 2021) analyzes Linux IoT malware using the MITRE ATT&CK framework, focusing on specific families and comparing the techniques employed. While it applies static and dynamic analysis methods, its methodology is less comprehensive than ours, especially in terms of scale and behavioral diversity.

In terms of macOS malware research, notable work includes (Pham et al., 2019), which presents a framework specifically designed to analyze macOS malware and resist anti-evasion techniques. Published in 2019, this framework supports macOS versions from 10.6 Snow Leopard to 10.12 Sierra, the latter released in 2016. Another work focuses on the use of memory forensics and a newly developed Volatility plugin (Manna et al., 2021), which aims to identify Objective-C-based malware on macOS through memory forensics techniques (Case and Richard, 2016). Additionally, the work in (Kowalczyk et al., 2024) provides a technical analysis along with mitigation recommendations, specifically focused on macOS ransomware. Similarly, the work in (Thaeler et al., 2023) aims to improve macOS malware detection using machine learning, incorporating string analysis and Mach-O static analysis. This research compares attributes of non-malicious and malicious samples. Some of the most comprehensive analysis of macOS malware can be found in (Wardle, 2022), which provides an extensive and detailed guide on how macOS malware should be analyzed and how it behaves. Our analysis aligns with the behavioral trends of this work, but quantifies them on a much larger scale, confirming that information gathering and C2 remain predominant tactics.

Compared to similar studies on different platforms, our dataset is of comparable (or even greater) magnitude. For instance, Cozzi et al. (2018) analyzed 10,548 Linux malware samples using both static and dynamic methods. Regarding Windows, Oosthoek and Doerr (2019) examined 951 binaries (one for each malware family in Malpedia), and the longitudinal study carried in (Botacin et al., 2021a) analyzed 41,084 Windows samples. Similarly, Ugarte-Pedrero et al. (2019) processed 172,000 Windows executables, collected in a single day in a private, isolated environment. Furthermore, the IoT-focused study provided in (Chierzi, Veronica and Mercês, Fernando, 2021) examined only 14 samples.

In essence, previous works on macOS malware has primarily focused on developing tools, detection frameworks, or small-scale behavioral analysis. In contrast, our work complements these efforts by validating and expanding their observations on a much larger empirical basis. Several patterns reported by Wardle (2022) and Thaeler et al. (2023), such as the prevalence of persistence and C2 mechanisms, are quantitatively confirmed in our dataset. Conversely, other behaviors, such as lateral movement, appear to be much less common. This comparison not only supports the validity of our results but also situates them within the broader evolution of the macOS threat landscape.

9. Conclusions and Future Work

In this paper, we presented a detailed analysis of macOS malware by filtering and examining a large collection of Mach-O samples through both static and dynamic analysis techniques. We provided an overview of the main tactics and techniques used to target macOS environments, structured according to the MITRE ATT&CK framework and focusing on the malware attack cycle phases.

Our findings highlight the critical role of techniques related to information gathering (discovery, collection), exfiltration, and command and control within the current macOS malware landscape. We hope that this work contributes to a deeper understanding of macOS malware and serves as a foundation for future research, as well as the development of more robust defense strategies tailored to the unique challenges of macOS environments.

As future work, one promising avenue we plan to explore is predictive modeling and behavioral classification. By leveraging the behavioral artifacts identified in our work as high-fidelity features, we aim to design machine learning algorithms capable of automatically classifying new malware samples. Furthermore, we are currently expanding this work by malware lineage and code evolution analysis. By applying code similarity techniques and function-level binary comparisons to our large-scale dataset, we seek to identify evolutionary relationships among malware families. This analysis will allow us to trace variant derivation and highlight malicious code reuse across the macOS ecosystem.

Acknowledgments

This research was supported in part by grant PID2023-151467OA-I00 (CRAPER), funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU, by grant TED2021-131115A-I00 (MIMFA), funded by MICIU/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR, by grants *Proyecto Estratégico Ciberseguridad EINA UNIZAR* and *Cátedra Internacional de Ciberseguridad UNIZAR*, funded by the Spanish National Cybersecurity Institute (INCIBE) and the European Union NextGenerationEU/PRTR, by grant *Programa de Proyectos Estratégicos de Grupos de Investigación (DisCo, ref. T21-23R)*, funded by the University, Industry and Innovation Department of the Aragonese Government.

Declaration of Generative AI and AI-Assisted Technologies in the Writing Process

During the preparation of this work, the authors used ChatGPT-4 to improve readability and language. After using this tool/service, the authors reviewed and edited the content as needed and assume full responsibility for the content of the publication.

References

- , 2022. Detect-It-Easy. [Online; <https://github.com/horsicq/Detect-It-Easy>]. Accessed on Sep 30, 2024.
- , 2024. MalwareBazaar – Malware sample exchange. [Online; <https://bazaar.abuse.ch/>]. Accessed on Sep 4, 2024.
- , 2024. VirusTotal. [Online; <https://www.virustotal.com/>]. Accessed on Sep 4, 2024.
- Acar, Y., Backes, M., Bugiel, S., Fahl, S., McDaniel, P., Smith, M., 2016. SoK: Lessons Learned from Android Security Research for Appified Software Platforms, in: 2016 IEEE Symposium on Security and Privacy (SP), pp. 433–451.
- Afianian, A., Niksefat, S., Sadeghiyan, B., Baptiste, D., 2019. Malware Dynamic Analysis Evasion Techniques: A Survey. *ACM Comput. Surv.* 52.
- Al Alsadi, A.A., Sameshima, K., Bleier, J., Yoshioka, K., Lindorfer, M., van Eeten, M., Gañán, C.H., 2022. No Spring Chicken: Quantifying the Lifespan of Exploits in IoT Malware Using Static and Dynamic Analysis, in: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security, Association for Computing Machinery, New York, NY, USA, p. 309–321.
- Alrawi, O., Lever, C., Valakuzhy, K., Court, R., Snow, K., Monrose, F., Antonakakis, M., 2021. The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle, in: 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, pp. 3505–3522.
- Apple Media Helpline, 2006. Apple to Use Intel Microprocessors Beginning in 2006. [Online; <https://www.apple.com/newsroom/2005/06/06Apple-to-Use-Intel-Microprocessors-Beginning-in-2006/>]. Accessed on November 9, 2024.
- Apple Media Helpline, 2020. Apple announces Mac transition to Apple silicon. [Online; <https://www.apple.com/newsroom/2020/06/apple-announces-mac-transition-to-apple-silicon/>]. Accessed on November 9, 2024.
- Apple Support, . Building a Universal macOS Binary. [Online; <https://developer.apple.com/documentation/apple-silicon/building-a-universal-macos-binary>]. Accessed on November 1, 2024.
- Apple Support, 2021. Startup security in macOS. [Online; <https://support.apple.com/en-gb/guide/deployment/dep5810e849c/web>]. Accessed on November 1, 2024.
- Apple Support, 2023. About System Integrity Protection on your Mac. [Online; <https://support.apple.com/en-us/102149>]. Accessed on November 1, 2024.
- Apple Support, 2024a. App Sandbox. [Online; <https://developer.apple.com/documentation/security/app-sandbox>]. Accessed on November 1, 2024.
- Apple Support, 2024b. Protecting against malware in macOS. [Online; <https://support.apple.com/en-gb/guide/security/sec469d47bd8/web>]. Accessed on November 1, 2024.
- Apple Support, 2024c. Secure Enclave. [Online; <https://support.apple.com/en-gb/guide/security/sec59b0b31ff/web>]. Accessed on November 1, 2024.
- Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., Kruegel, C., 2009. A view on current malware behaviors, in: Proceedings of the 2nd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More, USENIX Association, USA, p. 8.
- Botacin, M., Aghakhani, H., Ortolani, S., Kruegel, C., Vigna, G., Oliveira, D., Geus, P.L.D., Grégio, A., 2021a. One Size Does Not Fit All: A Longitudinal Analysis of Brazilian Financial Malware. *ACM Trans. Priv. Secur.* 24.
- Botacin, M., Ceschin, F., Sun, R., Oliveira, D., Grégio, A., 2021b. Challenges and pitfalls in malware research. *Computers & Security* 106, 102287.
- Calleja, A., Tapiador, J., Caballero, J., 2019. The MalSource Dataset: Quantifying Complexity and Code Reuse in Malware Development. *IEEE Transactions on Information Forensics and Security* 14, 3175–3190. doi:10.1109/TIFS.2018.2885512.
- Case, A., Richard, G.G., 2016. Detecting objective-C malware through memory forensics. *Digital Investigation* 18, S3–S10.
- Cheng, B., Ming, J., Fu, J., Peng, G., Chen, T., Zhang, X., Marion, J.Y., 2018. Towards Paving the Way for Large-Scale Windows Malware Analysis: Generic Binary Unpacking with Orders-of-Magnitude Performance Boost, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Association for Computing Machinery, New York, NY, USA, p. 395–411.

- Chierzi, Veronica and Mercês, Fernando, 2021. Evolution of IoT Linux Malware: A MITRE ATT&CK TTP Based Approach, in: 2021 APWG Symposium on Electronic Crime Research (eCrime).
- Corvus Forensics, 2024. VirusShare.com - Because Sharing is Caring. [Online; <https://virusshare.com/>]. Accessed on Sep 4, 2024.
- Cozzi, E., Graziano, M., Fratantonio, Y., Balzarotti, D., 2018. Understanding Linux Malware, in: 2018 IEEE Symposium on Security and Privacy (SP), pp. 161–175.
- Daniel Lastanao Miró, Javier Carrillo-Mondéjar, R.J.R., 2025. Dataset for macOS malware analysis. URL: <https://doi.org/10.5281/zenodo.16900075>, doi:10.5281/zenodo.16900075.
- DeveloperApple, 2014. Working with Blocks. [Online; <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithBlocks/WorkingwithBlocks.html>]. Accessed on July 2, 2025.
- DGArchive, 2025. DGArchive. [Online; <https://dgarchive.caad.fkie.fraunhofer.de/>]. Accessed on July 7, 2025.
- Filho, A.S., Rodríguez, R.J., Feitosa, E.L., 2022. Evasion and Countermeasures Techniques to Detect Dynamic Binary Instrumentation Frameworks. *Digital Threats: Research and Practice* 3, 28.
- Galloro, N., Polino, M., Carminati, M., Continnella, A., Zanero, S., 2022. A Systematical and longitudinal study of evasive behaviors in windows malware. *Computers & Security* 113, 102550.
- HackTricks, 2024. macOS Code Signing. [Online; <https://book.hacktricks.wiki/es/macOS-hardening/macOS-security-and-privilege-escalation/macOS-security-protections/macOS-code-signing.html>]. Accessed on June 3, 2025.
- Intel471, 2024. MacOS is Increasingly Targeted by Threat Actors. [Online; <https://intel471.com/blog/macOS-is-increasingly-targeted-by-threat-actors/>]. Accessed on November 1, 2024.
- Jamf, 2021. Shlayer malware abusing Gatekeeper bypass on macOS. [Online; <https://www.jamf.com/blog/shlayer-malware-abusing-gatekeeper-bypass-on-macos/>]. Accessed on July 22, 2025.
- Kandji, 2024. 2024 Apple in the Enterprise Report. [Online; <https://www.kandji.io/apple-in-the-enterprise-report/>]. Accessed on November 1, 2024.
- Kowalczyk, H., Zieliński, P., Nowak, A., et al., 2024. Dissecting MacOS Ransomware: A Comparative Analysis and Mitigation Strategies. *resreport. DigitPolska Sp. z o.o.* doi:10.21203/rs.3.rs-4385485/v1.
- Levin, J., 2016. MacOS and iOS Internals, Volume III: User Mode. 2nd ed., TechnoGeeks Press.
- Levin, J., 2017. MacOS and iOS Internals, Volume I: User Mode. 2nd ed., TechnoGeeks Press.
- Ling, X., Wu, L., Zhang, J., Qu, Z., Deng, W., Chen, X., Qian, Y., Wu, C., Ji, S., Luo, T., Wu, J., Wu, Y., 2023. Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art. *Computers & Security* 128, 103134.
- Malwarebytes, 2024. 2024 State of Malware. [Online; <https://www.threatdown.com/dl-state-of-malware-2024/>]. Accessed on November 1, 2024.
- Manna, M., Case, A., Ali-Gombe, A., Richard, G.G., 2021. Modern macOS userland runtime analysis. *Forensic Science International: Digital Investigation* 38, 301221.
- Microsoft, 2022. PE Format. [Online; <https://learn.microsoft.com/en-us/windows/win32/debug/pe-format>]. Accessed on October 07, 2022.
- MITRE Corporation, 2024. MITRE ATT&CK – macOS Matrix v16.0. [Online; <https://attack.mitre.org/matrices/enterprise/macOS/>]. Accessed on November 1, 2024.
- Moussaileb, R., Cuppens, N., Lanet, J.L., Boudier, H.L., 2021. A Survey on Windows-based Ransomware Taxonomy and Detection Mechanisms. *ACM Comput. Surv.* 54.
- Naik, N., Jenkins, P., Cooke, R., Gillett, J., Jin, Y., 2020. Evaluating Automatically Generated YARA Rules and Enhancing Their Effectiveness, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1146–1153.
- NeutrinoAPI, 2025. NeutrinoAPI. [Online; <https://www.neutrinoapi.com/>]. Accessed on August 4, 2025.
- Oosthoek, K., Doerr, C., 2019. SoK: ATT&CK Techniques and Trends in Windows Malware, in: Chen, S., Choo, K.K.R., Fu, X., Lou, W., Mohaisen, A. (Eds.), *Security and Privacy in Communication Networks*, Springer International Publishing, Cham. pp. 406–425.
- Pham, D.P., Vu, D.L., Massacci, F., 2019. Mac-A-Mal: macOS malware analysis framework resistant to anti evasion techniques. *Journal of Computer Virology and Hacking Techniques* 15, 249–257.
- Qamar, A., Karim, A., Chang, V., 2019. Mobile malware attacks: Review, taxonomy & future directions. *Future Generation Computer Systems* 97, 887–909.
- Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., Xiang, Y., 2020. A Survey of Android Malware Detection with Deep Neural Models. *ACM Comput. Surv.* 53.
- Radare2 Team, 2017. Radare2 Book. GitHub.
- Sebastián, S., Caballero, J., 2020. AVclass2: Massive Malware Tag Extraction from AV Labels, in: *Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC)*, Association for Computing Machinery.
- Tekiner, E., Acar, A., Uluagac, A.S., Kirda, E., Selcuk, A.A., 2021. SoK: Cryptojacking Malware, in: 2021 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 120–139.
- Thaaler, A., Yigit, Y., Maglaras, L., Buchanan, W.J., Moradpoor, N., Russell, G., 2023. Enhancing Mac OS Malware Detection through Machine Learning and Mach-O File Analysis, in: 2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), pp. 170–175.
- Ugarte-Pedrero, X., Graziano, M., Balzarotti, D., 2019. A Close Look at a Daily Dataset of Malware Samples. *ACM Trans. Priv. Secur.* 22.
- VirusTotal, 2025. External behavioural engines sandboxes. [Online; <https://docs.virustotal.com/docs/external-sandboxes>]. Accessed on April 2, 2025.
- Wang, J., Wang, L., Dong, F., Wang, H., 2023. Re-measuring the Label Dynamics of Online Anti-Malware Engines from Millions of Samples, in: *Proceedings of the 2023 ACM on Internet Measurement Conference*, Association for Computing Machinery, New York, NY, USA. pp. 253–267.
- Wardle, P., 2022. *The Art of Mac Malware: The Guide to Analyzing Malicious Software*. 1st ed., No Starch Press.
- Yong Wong, M., Landen, M., Antonakakis, M., Blough, D.M., Redmiles, E.M., Ahamad, M., 2021. An Inside Look into the Practice of Malware Analysis, in: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, New York, NY, USA. p. 3053–3069.
- Zeller, A., 2003. Program Analysis: A Hierarchy, in: *Proceedings of the Workshop on Dynamic Analysis (WODA 2003)*, pp. 1–4.
- Zhu, S., Shi, J., Yang, L., Qin, B., Zhang, Z., Song, L., Wang, G., 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines, in: 29th USENIX Security Symposium (USENIX Security 20), USENIX Association. pp. 2361–2378.

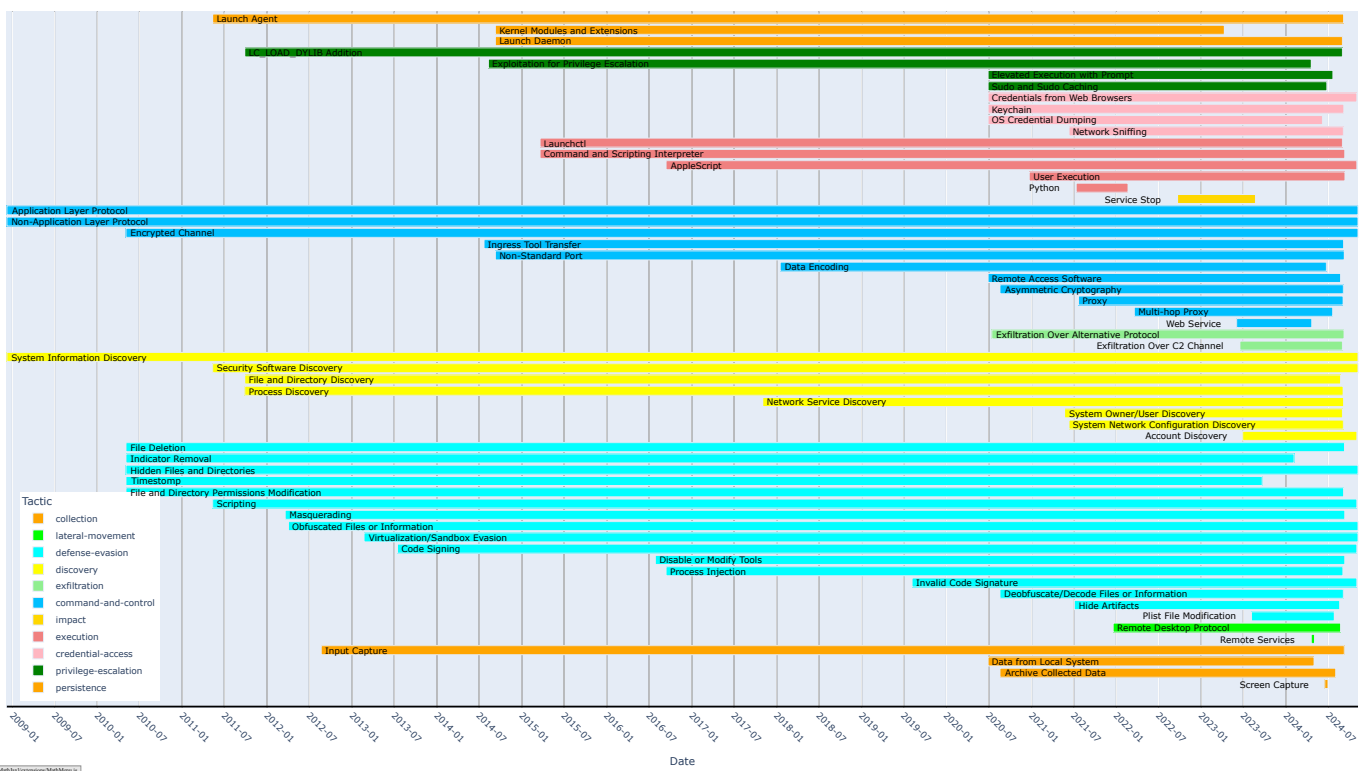


Figure 8: Timeline of macOS malware techniques by tactic, illustrating their evolution over time.