CRIBA: A Tool for Comprehensive Analysis of Cryptographic Ransomware's I/O Behavior

Tânia Esteves
INESC TEC & U. Minho
tania.c.araujo@inesctec.pt

Bruno Pereira
INESC TEC & U. Minho
bruno.f.pereira@inesctec.pt

Rui Pedro Oliveira
INESC TEC & U. Minho
rui.v.oliveira@inesctec.pt

João Marco
INESC TEC & U. Minho
joao.marco@inesctec.pt

João Paulo
INESC TEC & U. Minho
jtpaulo@inesctec.pt

2023 42nd International Symposium on Reliable Distributed Systems (SRDS) | 979-8-3503-2910-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/SRDS60354.2023.00015

***Abstract*—Cryptographic ransomware attacks are constantly evolving by obfuscating their distinctive features (*e.g.*, I/O patterns) to bypass detection mechanisms and to run unnoticed at infected servers. Thus, efficiently exploring the I/O behavior of ransomware families is crucial so that security analysts and engineers can better understand these and, with such knowledge, enhance existing detection methods. In this paper, we propose CRIBA, an open-source framework that simplifies the exploration, analysis, and comparison of I/O patterns for Linux cryptographic ransomware. Our solution combines the collection of comprehensive information about system calls issued by ransomware samples, with a customizable and automated analysis and visualization pipeline, including tailored correlation algorithms and visualizations. Our study, including 5 Linux ransomware families, shows that CRIBA provides comprehensive insights about the I/O patterns of these attacks while aiding in exploring common and differentiating traits across families.**

***Index Terms*—cryptographic ransomware, I/O tracing and analysis, operating systems**

I. Introduction

Cryptographic ransomware is one of the most well-known and damaging types of malware, which acts by encrypting data at infected servers and then demanding a ransom in exchange for the cryptographic key necessary to decrypt compromised data to its original format [1], [2]. This malicious software is now spread across distinct operating systems (OSs) other than Windows, such as Android and Linux. As the latter OS is typically used by large institutions (*i.e.*, governments, companies) holding critical and private information, ransomware attacks on their distributed infrastructures can have devastating effects, as observed for the Colonial Pipeline, Quanta Computer, and Konica Minolta attacks in 2021 [3]–[5].

Current ransomware detection and prevention tools are mostly based on classification and machine learning algorithms that can differentiate between malign and benign applications through key features, such as unique I/O patterns exhibited by ransomware attacks (*e.g.*, targeted files, API call patterns) [6]–[8]. These features are distilled from information collected statically from binary inspection or dynamically from observing the I/O interaction of ransomware samples (*i.e.*, binaries) with the OS. The latter is necessary for samples using concealment techniques (*i.e.*, obfuscation, polymorphism, encryption) that make binary inspection inefficient [2], [9].

However, given the sheer amount of cryptographic ransomware families, which are constantly evolving to bypass detection mechanisms, it becomes challenging to understand what I/O information needs to be collected dynamically at runtime and used as a feature for classification purposes. For this, security analysts and engineers require deep knowledge of how ransomware families work and evolve over time.

Frameworks, such as Cuckoo [10] and Limon [11], provide sandbox environments to run ransomware samples and output to users summaries of their suspicious activities. Since these summaries are limited in terms of analysis scope, these frameworks also output detailed execution logs (traces), which are crucial for an in-depth exploration of ransomware's I/O patterns. However, we argue that this is a sub-optimal approach as it leaves to users the inspection of large traces (*i.e.*, containing thousands to millions of I/O events), which could be automated with proper analysis and visualization tools.

Therefore, while focusing on the Linux OS, the main insight of this paper is that to further comprehend the I/O behavior of ransomware, one should leverage all the information collected during its execution and automate its analysis and visualization. Namely, by exploring the system calls (syscalls) used by a ransomware sample, their arguments, and their contextual information (*e.g.*, process name, process ID, thread ID), one would be able to obtain detailed information about the sample. For example, it would be possible to *i)* observe how many processes/threads are created and learn about their specialization (*e.g.*, encrypt files, write ransom notes); *ii)* understand how the infected file system is transversed and which files are being targeted; *iii)* learn about specific I/O patterns done over infected files (*e.g.*, encryption key generation and persistence, file's extension renaming, targeted file offsets).

While this information is useful for analysts to better understand ransomware attacks, compare distinct families and observe their I/O patterns' evolution, to implement such an idea, one must address the following challenges:

(A) *Non-intrusive and dynamic data collection*: Most of the available ransomware samples are packed binaries whose

2575-8462/23/$31.00 ©2023 IEEE
DOI 10.1109/SRDS60354.2023.00015

source code is undisclosed. Thus, the collection of relevant information (*tracing*) should be *dynamic* (*i.e.*, done along with the sample's execution) and *non-intrusive* (*i.e.*, without requiring modifications to its source code).

(B) *Comprehensive data*: To enable a comprehensive exploration and analysis of ransomware's I/O behavior, collected traces must include diverse and detailed information regarding the sample's interaction with the OS (*e.g.*, process creation, syscall types, arguments, and context). Such information can also be complemented with other system metrics (*e.g.*, CPU usage).

(C) *Integrated analysis and visualization*: The analysis pipeline should be integrated with the data collection phase while efficiently handling large volumes of traced data (*i.e.*, thousands to millions of I/O events), allowing its storage, processing, and visualization.

(D) *Automated analysis*: Manually exploring traced data and finding the most appropriate queries to observe specific I/O patterns is a complex and time-consuming task, which could be automated with tailored correlation algorithms.

(E) *Pre-defined Visualizations*: Users should have access to pre-defined and customizable visualizations that provide a human-readable and explainable way of exploring, understanding, and summarizing the analysis findings.

(F) *Comparision between families*: To efficiently compare the I/O behavior of different ransomware families, one must store and index the data traced for each sample, correlate such information, and create adequate visualizations that help pinpoint their main similarities and differences.

The previous challenges are addressed with CRIBA, a tool for capturing, analyzing, and visualizing the behavior of Linux cryptographic ransomware. Briefly, CRIBA uses dynamic analysis for collecting information about I/O syscalls issued by ransomware samples along with other system metrics. Also, it provides an integrated analysis and visualization pipeline that employs several correlation algorithms and combines these with custom visual representations to enable comprehensive insights about the I/O patterns of ransomware attacks. In more detail, the paper provides the following contributions:

- An open-source tool integrating the collection, analysis, and visualization of execution traces from ransomware.
- Support for the collection and analysis of comprehensive information about I/O syscalls (*e.g.*, type, arguments), their contextual information (*e.g.*, process ID, offset), and relation with other system metrics (*e.g.*, CPU).
- Automated analysis capabilities through 6 algorithms that ease the study and comparison of ransomware samples by pinpointing their file system transversal, file access, and file extension manipulation patterns.
- A pre-defined set of visualizations, organized into 7 distinct dashboards, for summarizing and exploring collected information, and the outputs of the correlation algorithms, in a human-readable and explainable fashion.
- A comprehensive analysis and comparison of 5 Linux ransomware families that shows CRIBA's capabilities.

The conducted experimental study shows that CRIBA automates the analysis and observation of generic behavior from ransomware samples (*e.g.,* the number of processes, type of syscalls, file system transversal). Further, it enables the analysis and comparison of intrinsic and complex I/O behavior (*e.g.*, file access patterns, extension manipulation) related to the creation of ransom notes, file encryption, and evasion techniques used by each family. All artifacts discussed in the paper, including CRIBA, datasets, scripts, and the corresponding analysis and visualization outputs, are publicly available at https://github.com/dsrhaslab/criba.

II. Background

We now overview the workflow of cryptographic ransomware while highlighting some of its unique I/O features.

A. Main phases of ransomware attacks

Cryptographic ransomware typically acts in four phases [12]. First, the attacker exploits system vulnerabilities (*e.g.*, kernel bugs) or uses social engineering techniques (*e.g.*, phishing emails) to install a malicious sample at the victim's machine(s) (*Infection* phase). Once installed and running, the sample establishes a connection with its Command and Control (C&C) server to retrieve necessary information for data encryption (*e.g.*, encryption keys) and/or exfiltrate information about the infected system (*e.g.*, hostname, hardware info) to the attacker (*Communication with C&C servers* phase). Then, the ransomware transverses the files at the infected server(s) and encrypts their data, blocking access to these (*Destruction* phase). In the end, the ransomware leaves a ransom note informing the victim about the attack and disclosing payment instructions (*Extortion* phase).

B. Data encryption

Cryptographic ransomware typically follows a hybrid approach combining symmetric and asymmetric encryption schemes [12]. Namely, it starts by locally creating a symmetric key, usually a different key per infected file. It then reads the file's content and encrypts it with the generated key. Symmetric key encryption schemes impose lower CPU load and have faster encryption times than asymmetric ones [13].

To prevent the victim from discovering symmetric keys and recovering the original files' content, the ransomware encrypts these with the attacker's public key obtained, for instance, during the *Communication with C&C servers* phase. The encrypted file's data and corresponding encrypted symmetric key are both written to the targeted file, which is usually renamed to include a new extension (*e.g.*, .ecrypt in EREBUS). Thus, to recover the original files, the victim first needs to obtain the attacker's private key and then use it to decrypt the symmetric keys needed for the files' data decryption.

C. Detection features

The ransomware actions to encrypt the victim's files result in intensive I/O patterns with several characteristics that deviate from the normal behavior of benign applications:

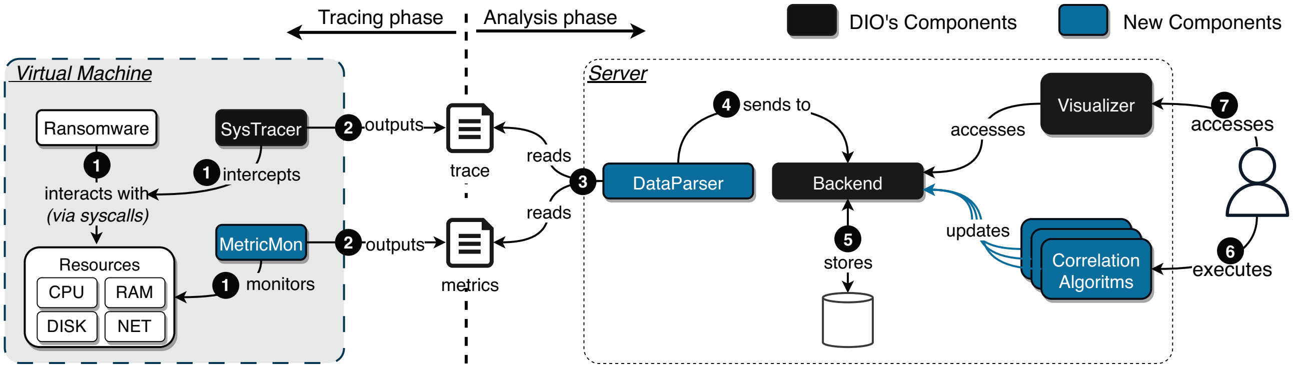


Fig. 1: CRIBA's design and flow of events for the tracing and analysis phases.

- *Directory search* - typically, all directories at the infected machine are transversed in search for files to encrypt [7].
- *Files data access* - encryption usually requires rewriting the whole file's data in a short time window [6].
- *Number of storage operations* - encrypting several files results in a significant amount of storage I/O operations, such as opening, reading, writing, and closing each file.
- *Unknown file extensions* - by changing the extension of encrypted files, ransomware samples execute an abnormal number of rename operations. Moreover, this results in the appearance of new and unknown file extensions [7].
- *CPU usage* - by encrypting all files, the sample imposes a high CPU load on the victim's machine [13].
- *Network communication* - the communication with the C&C usually translates into network operations targeting unknown network domains [8].

These are some of the features that detection tools use to identify the malicious activity of cryptographic ransomware.

D. Evasion techniques

Some families use evasion techniques to retard or avoid being detected. For instance, many families include the public encryption key within the binary to avoid communication with the C&C [13]. Other families reduce CPU load and encryption time, and hide I/O patterns by encrypting only a subset of files on the infected machine. File selection can be based on *i)* its extension (e.g., work-related documents such as *.pdf*, *.docx*, *.txt*; or VM-related files such as *vmdk*, *.vmem*, *.vswp*); *ii)* its size (*e.g.*, larger files will most probably contain important data); or *iii)* arbitrary. Some families also limit the number of bytes to encrypt in each file, which is usually sufficient to avoid recovering their full content.

With CRIBA, we aim to assist security analysts in exploring and analyzing the I/O behavior of cryptographic ransomware samples to understand better how they operate, identify and refine key features for their detection, and learn about their techniques used to evade detection tools.

III. Design

Figure 1 depicts CRIBA's architecture, which is built on top of DIO, a generic tool for observing and diagnosing the I/O interactions between applications and in-kernel POSIX storage systems [14]. DIO provides modules for *i)* collecting information about applications' syscalls without requiring access or modification to their source code (*challenge* §I-(A)); and *ii)* storing, analyzing and visualizing the collected information (*challenge* §I-(C)). Next, we overview DIO and highlight the novel contributions introduced by CRIBA.

A. DIO

DIO's design consists of a *tracer* component, which runs collocated with the application being traced and records information about its syscalls, and an *analysis pipeline*, which may be deployed on a different server(s) and that allows storing, querying, and visualizing the collected information.

The *tracer* uses the eBPF technology to non-intrusively intercept storage-related syscalls made by the application. In addition to collecting information about the syscalls (*i.e.*, type, arguments, and return value), DIO's *tracer* also collects contextual information (*i.e.*, timestamps, process, and thread IDs, process name) and additional information from the kernel, such as file types, file paths, and file offsets. Further, by parsing and sending traced data directly to the *analysis pipeline* as soon as it is collected, DIO provides near real-time analysis.

The *analysis pipeline* includes two main components. The *backend* receives data collected by the *tracer*, indexes it, and provides access to it through a querying API. The *visualizer* is connected with the *backend* and provides users with the mechanisms to summarize the data through customizable visualizations (*e.g.*, tables, histograms, time series plots).

B. CRIBA*'s design*

Although DIO's analysis pipeline is useful for observing the I/O behavior of applications, it still requires users to spend significant time exploring and manually building/customizing queries and visualizations to analyze collected traces.

CRIBA extends DIO to collect more information at the tracing phase, including network operations and system's resources metrics (*challenge* §I-(B)), and to automate the analysis process by providing a set of correlation algorithms and pre-defined visualizations (*challenges* I-(D),(E)) tailored for the exploration of cryptographic ransomware. Next, we highlight the modifications made to DIO's original design and describe the new components added (blue boxes at Figure 1).

***SysTracer*.** DIO's *tracer* is focused on the interception of storage-related sycalls. To consider also network-related requests, we modified this module to intercept 13 more syscalls (*e.g.*, `connect`, `accept`, `send`, `receive`). The full set of supported syscalls is shown in Table V of Appendix A.

***MetricMon*.** To obtain information about the system's resource usage, we introduced a new module for collecting statistics, including CPU, memory, and disk usage.

***DataParser*.** Ransomware samples must run in a controlled environment (*e.g.*, isolated VM) to avoid infecting the experimental servers. Thus, *SysTracer* and *MetricMon* were configured to save collected information to disk instead of sending it directly to the analysis pipeline. Further, we introduced the *DataParser* module that parses the trace files and forwards these to the analysis pipeline.

***Backend and correlation algorithms*.** The *Backend* component, which may be deployed on separate server(s), is identical to the one offered in DIO and provides the functionalities of storage and exploration of collected information. However, to ease the analysis of cryptographic ransomware, we developed several new correlation algorithms that query the *Backend*, analyze and correlate queried data, and send the analysis results back to the *Backend*. As further explained in §III-D and demonstrated in §V, these algorithms provide relevant information to understand how ransomware behaves and to find interesting and distinctive I/O patterns. Further, our design is extensible, enabling users to develop other correlation algorithms that may suit their analysis goals.

***Visualizer*.** The *Visualizer* component, which is also based on DIO, was extended to include visual representations tailored for observing ransomware's analysis findings (*e.g.*, for the output of correlation algorithms) thus, simplifying users' exploration and making it more explainable. Particularly, we built several dashboards that allow observing general statistics about the traced execution (*e.g.*, execution time, number of processes and threads, number and type of syscalls) and explore each of these in more detail (*e.g.*, most accessed files, file system transversal, sequence of syscalls per files). Each dashboard is mentioned explicitly in §V while discussing the analysis findings contained in it. Also, our design is extensible as users can create new visualizations based on their analysis needs.

C. CRIBA*'s workflow*

As shown in Figure 1, CRIBA's components are executed in two separate phases: *i)* the tracing phase, where information from the ransomware execution is collected; and *ii)* the analysis phase, where collected data is analyzed and visualized.

In the tracing phase, the ransomware sample is executed in a controlled environment along with *SysTracer* that intercepts its I/O syscalls, and *MetricsMon* that monitors system statistics (❶). When the ransomware finishes its execution, these components' output files are extracted from the controlled environment (❷) to initiate the analysis process.

In the analysis phase, the *DataParser* is used to read the tracing output files (❸) and forward these to the *Backend* (❹). The latter persists and indexes collected data (❺) and provides access to it through a querying API. Meanwhile, users can execute the provided *correlation algorithms* (❻) and access the *Visualizer* dashboards (❼) to visually explore the output of these and other information contained at the indexed data.

D. *Correlation Algorithms*

Taking into account the ransomware features discussed in §II, we developed six algorithms that ease their analysis.

UNExt. The *UNExt* algorithm extracts the file name and extension from the file paths targeted by traced syscall events. Specifically, for every event accessing[1] a file path (*e.g.*, `read`, `write`, `stat`), the algorithm splits the full path (*e.g.*, `/files/example.txt`) to obtain the file name (*e.g.*, `example.txt`) and the file extension (*e.g.*, `.txt`). This algorithm is implemented as a search and update query that is fully executed at the *Backend*. With its output, users can explore, at the *Visualizer*, the file names and extensions accessed by ransomware samples.

DsetU. The *DsetU* algorithm compares the list of file paths accessed by the ransomware sample with the full list of file paths contained in a given dataset collection, provided as input by the user (*e.g.*, the experimental dataset described in §IV). The algorithm then outputs to the *Backend* which dataset's files and extensions were accessed by the ransomware. This output can then be explored with the *Visualizer* to uncover samples targeting specific files and extensions.

Transversals. The *Transversals* algorithm determines the order in which ransomware threads transverse the file system, namely Breadth First Search (BFS), Depth First Search (DFS), or unknown. Alg. III.1 shows the algorithm for identifying DFS. By analyzing opened files, it builds a file tree (L2) and obtains an order – $dfsOrder$ – in which the files would have been visited with DFS (L3). Then it correlates the actual file opening order done by the thread with $dfsOrder$ (L4-L8). Next, it builds a SegmentTree structure [15] (L9) to efficiently verify if external files were visited while transversing a given subtree (L10-L15), meaning that the thread is not doing DFS. Note that the algorithm tolerates different access orders to files on the same directory (*e.g.*, alphabetical, creation time, *etc.*).

The search type done by each thread, along with the information containing opened files and folders, are sent to the *Backend* and can be explored at the *Visualizer* (*e.g.*, through a tree representation as shown in Figure 9 in Appendix D).

FnGram. The *FnGram* algorithm computes the n-grams collocations for the files accessed by a ransomware sample over time. Specifically, the algorithm queries the *Backend* to obtain

[1] In the paper, file access means at least one syscall is done over the file.

Algorithm III.1: DFS identification algorithm.

```
Input: Files opened over time ($fpaths$)
Output: True if DFS was observed
Function isDFS($fpaths$) is
    $tree \leftarrow$ buildFileTree($fpaths$)
    $dfsOrder \leftarrow$ transverseDFS($tree$)
    $list \leftarrow []$
    $i \leftarrow 0$
    for $file \in fpaths$ do
        $idx \leftarrow dfsOrder[file]$
        $list[idx] = i$
        $i+ = 1$
    $st \leftarrow$ buildSegmentTree($list$)
    for $file \in fpaths$ do
        $a \leftarrow st$.getMaxElemInsideSubtree()
        $b \leftarrow st$.getMinElemOusideSubtree()
        $st$.remove($file$)
        if $a > b$ then
            return $false$
    return $true$
```

Algorithm III.2: FSysSeq correlation algorithm.

```
  Input: Syscalls for a given file over time ($syscalls$)
  Output: Syscall sequence ($sequence$)
1 Function ComputeSysSeq($syscalls$) is
2     $sequence \leftarrow []$
3     $prev\_tag \leftarrow$ NULL
4     for $s \leftarrow syscalls$ do
5         $tag \leftarrow$ get_tag($s$)
6         if $prev\_tag = NULL$ or $prev\_tag \neq tag$ then
7             $sequence$.append($tag$)
8         $prev\_tag \leftarrow tag$
9     return $sequence$
```

a list of the file paths accessed by each thread. Then, it computes and sends back to the *Backend* the *bigrams*, *trigrams*, and *quadgrams* for that input list. With this information, it is possible to depict dependencies between file accesses (*e.g.*, the trigram (`A.txt`, `B.txt`, `C.txt`) shows that `B.txt` was accessed after `A.txt` and before `C.txt`).

FSysSeq. The *FSysSeq* algorithm computes the sequence of consecutive unique syscalls done by ransomware threads to all files. First, for each file, the algorithm queries the *Backend* to obtain the list of syscalls (sorted by time). As depicted in Alg. III.2, each syscall in the list is translated to a tag (L5) of two letters (according to Table V in Appendix A) to reduce the length of the final sequence, while subsequent syscalls of the same type are reduced to a single one (L6-L7). For example, the list [`open`, `read`, `read`, `lseek`, `write`, `write`, `close`, `rename`] is reduced to "`OP`→`RD`→`LS`→`WR`→`CL`→`RN`". With these sequences, one is able to learn how ransomware threads access files and observe similar patterns between files (*e.g.*, identical sequences for ransom notes as shown in §V-B).

tfidfFam. The *tfidfFam* algorithm eases the comparison between distinct ransomware families. Specifically, for each family, the algorithm requests from the *Backend* all the values observed for a given category (*e.g.*, syscall type). These values are passed as input to the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm. The latter is a feature selection technique, often used by ransomware detection tools [16]–[18], that outputs a numerical statistic showing the relevance of each value (*e.g.*, relevance of `read` syscalls for the Erebus sample). Then, the cosine similarity is applied to the output of TF-IDF to have a single metric of comparison between families according to their most relevant values.

By sending back to the *Backend* the TF-IDF output, we allow users to easily observe the most relevant values per family and for a given category. By sending the cosine similarity results, we allow users to understand how similar/distinct the ransomware families are. We apply this algorithm to three categories (*i.e.* syscall type, file paths, and file extensions).

E. Implementation

SysTracer extends DIO's tracer and is implemented in Go while resorting to the BPF Compiler Collection (BCC) framework through the *gobpf* lib (v0.2.0) to intercept applications' syscalls. The *Backend*, *Visualizer* and *MetricMon* components are provided by instances of Elasticsearch [19], Kibana [20], and Metricbeat [21], respectively. The *DataParser* and the *correlation algorithms* are implemented in Python and interact with the Elasticsearch instance (*i.e.*, index, update, and query). The dashboards with pre-defined visualizations are provided along with CRIBA and include representations (*e.g.*, Figure 4) developed using the Vega-Lite [22] grammar.

IV. Evaluation Methodology

Our experimental evaluation shows how CRIBA automates and eases the work for users when: *i)* exploring and understanding both general and specific behaviors exhibited by ransomware samples; and *ii)* comparing different families to find common and distinct patterns across them.

A. Ransomware families

The experiments consider 5 Linux ransomware families, which were chosen based on their popularity and distinct traits.

- Erebus emerged in 2016 and is known for infecting thousands of computers and servers. A notorious example is the attack on the Linux infrastructure of a South Korean web hosting company in 2017 [23].
- REvil is a high-profile ransomware family discovered in 2019 that reached its peak activity in 2021. It targeted both widely known public figures and companies, including Quanta Computer, a supplier of Apple [4].

TABLE I: Execution time, process creation, accessed files and issued syscalls statistics for the ransomware families.

Family	*Execution*	*Process*		*Accesses*			*Syscalls*			
	time (min)	*PIDs*	*TIDs*	*Paths*	*Extensions*	*Transversal*	*Types*	*Events*	*Data / Metadata*	*Storage / Network*
AvosLocker	1.481	1	2	11 646	3 044	DFS	8	134 985	34.13% / 65.87%	100% / 0%
RansomEXX	3.126	1	5	85 583	19 341	DFS	9	703 575	31.99% / 68.01%	100% / 0%
REvil	8.719	12	13	39 384	8 275	DFS	9	774 007	41.83% / 58.17%	100% / 0%
Erebus	10.361	3	12	107 307	8 482	-	17	1 257 238	26.86% / 73.14%	99.96% / 0.04%
Darkside	0.386	1	6	11 244	12	DFS	19	21 070	25.06% / 74.94%	99.79% / 0.21%

- AvosLocker released a Linux variant in 2021. This family has been targeting critical infrastructures in countries such as the US, Canada, and UK [24].
- RansomEXX is a recent ransomware targeting Linux infrastructures. The Texas Department of Transportation, Konica Minolta, and Scottish Mental Health Charity were attacked by this malware between 2020 and 2022 [5].
- Darkside emerged in 2020 and was used to launch a global campaign infecting targets in 15 countries and multiple industry sectors [25]. It is known for, in 2021, targeting Colonial Pipeline, a company responsible for half of the fuel supply of the US East Coast [3].

As some ransomware samples require defining the number of encryption threads and the targeted file system directory path, we configured all samples to use 1 encryption thread and to target the dataset discussed next. For Erebus, which does not allow specifying these two arguments, and for Darkside, which does not allow changing the number of threads, we used their default configurations. Table VI in Appendix C shows the SHA256 hashes and execution commands for each sample.

B. File dataset

As in previous work [26], [27], the *Impressions* framework [28] was used to generate a synthetic dataset exhibiting a statistically accurate file system image with realistic metadata and content. The dataset, with 9.4 GiB, includes 35 418 files, with sizes ranging from 0 B to 800 MiBs. Files are spread across 3 510 directories with an average tree depth of 12 levels.

Darkside encrypts only specific file extensions (*.vmem*, *.vswp*, *.log* and *.vmdk*) that are not generated with the *Impressions* framework. Thus, we developed a script to change the file names of some dataset files, considering both small and large-sized files, to include these. The final dataset used in the experiments has 8 267 unique file extensions. Further information about the distribution of files' sizes and extensions is shown in Figures 7 and 8 in Appendix B.

C. Experimental Setup

Our testbed includes two environments. The ransomware samples and CRIBA's *SysTracer* and *MetricsMon* are executed in a controlled environment. Namely, they run inside a Virtual Machine (VM) configured with 2 GiB of RAM, 2 CPU cores, and a disk partition of 64 GiB. The VM is deployed on a server equipped with an 8-core Intel i9-9880H, 16 GiB of memory, and a 500 GiB SSD NVMe. The host OS runs *macOS Big Sur 11.7.6* while the guest OS runs *Ubuntu 22.04 LTS* with kernel *5.15.0*. The VM image is reverted to a previous (and clean) snapshot every time a ransomware sample is executed.

TABLE II: Top 3 syscall types issued per ransomware family.

Syscall	*Family*				
	AvosLocker	RansomEXX	REvil	Erebus	Darkside
#1	`lseek` (23.942%)	`lseek` (20.280%)	`read` (28.385%)	`read` (19.013%)	`stat` (53.711%)
#2	`read` (20.589%)	`write` (16.340%)	`lseek` (20.084%)	`stat` (15.871%)	`read` (10.332%)
#3	`fstat` (16.755%)	`read` (15.648%)	`close` (14.226%)	`openat` (14.671%)	`writev` (10.190%)

CRIBA's *Backend* and *Visualizer* components, as well as the correlation algorithms, run at the analysis environment, which consists of a separate server equipped with a 6-core Intel i5-9500, 16 GiB of memory and a 250 GiB NVMe SSD, and running *Ubuntu 20.04 LTS* with kernel *5.4.0*.

The execution time for the full analysis workflow, including the tracing, pre-processing, and loading of traced data (using CRIBA's *DataParser*), and execution of all correlation algorithms, took, on average, ≈19 minutes per family.

V. Evaluation Results

We now discuss the main analysis findings and takeaways from the experimental study. We start by introducing general statistics for the different families (§V-A), which are then further explored and justified with a more in-depth observation of specific features of ransomware samples, including their ransom notes creation (§V-B), data encryption (§V-C) and evasion techniques (§V-D). Finally, we highlight the main similarities and differences found across families (§V-E).

A. Overview

Tables I and II show general statistics provided by CRIBA for the 5 ransomware families considered in our experiments.

Observation 1. Families exhibit significantly different execution times, with Darkside running in less than 1 minute and Erebus taking more than 10 minutes.

Observation 2. Most families use a single process, except for Erebus (3 processes) and REvil (12 processes). The number of threads ranges from 2 in AvosLocker to 13 in REvil.

Observation 3. AvosLocker and Darkside access fewer unique file system paths and file extensions than the other samples. Erebus has the highest number of file accesses, while RansomEXX accesses the highest number of distinct file extensions. When considering only accesses to the dataset's

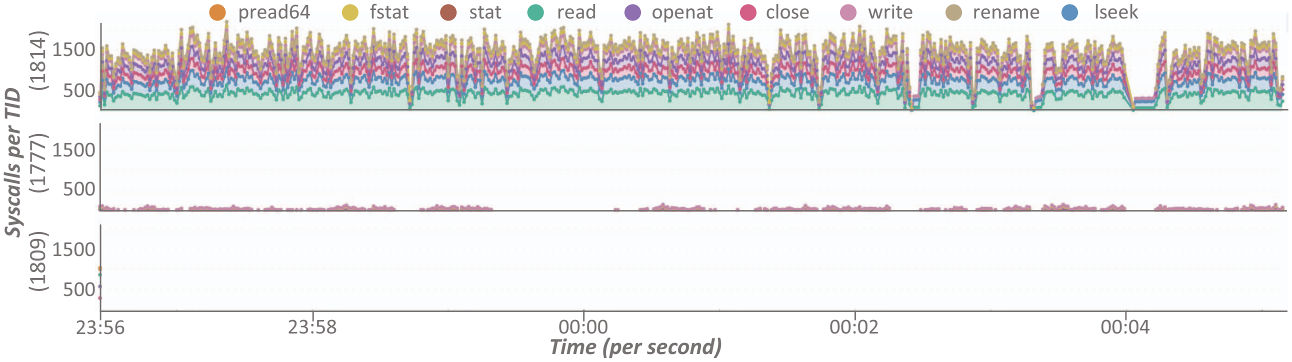


Fig. 2: Aggregated number of operations, separated by syscall type, for three distinct threads launched by REvil.

files, AvosLocker and Darkside only access ≈30% of these, while RansomEXX accesses almost 93%. REvil and Erebus are the only ones accessing all files in the dataset.

Observation 4. Except for Erebus, all families perform a depth-first search (DFS) to transverse the dataset's directory tree. This search is done by 2 threads in AvosLocker, REvil and Darkside, and 5 threads in RansomEXX. Also, all samples access system directories besides the dataset's ones (*e.g.*, `/usr`, `/proc`, `/dev`).

Observation 5. Darkside uses a wider range of different syscall types (*e.g.*, `stat`, `read`, `writev`) but performs fewer operations in total than the other families. AvosLocker issues fewer types of syscalls, while Erebus performs more than 1 million I/O operations.

Observation 6. The majority of issued syscalls are metadata-related. Namely, `lseek`, `stat`, and `fstat` are widely used by all families. In Darkside, half of the total issued I/O requests correspond to `stat` syscalls.

Observation 7. Erebus and Darkside are the only samples issuing network-related syscalls, such as `connect` or `recvfrom`. This indicates that these samples may be the only ones communicating with C&C servers.

Observation 8. For all families, the distribution of I/O load (*i.e.*, amount of requests) per thread varies. For instance, when considering the 13 threads created by REvil, only two do syscalls throughout the whole execution. Moreover, as depicted in Figure 2, most I/O requests are done by a single thread (TID 1814 issues 98.027% of the syscalls, while TID 1777 executes only 1.822%). The remaining threads perform fewer I/O requests and only at the beginning of the sample's execution (*e.g.*, TID 1809 does 0.132% of the syscalls).

> **Takeaways.** *General statistics show that ransomware families exhibit different patterns in terms of execution time, process and thread creation, and accessed files and extensions. Metadata-related storage operations are the most predominant type of issued syscall, while network-related calls are only issued by a few families. Most families transverse the file system in a DFS fashion.*

TABLE III: Syscall sequences for ransom notes per family.

Family	*Syscall sequence*
AvosLocker	*README_FOR_RESTORE*: `OP→ST→WR→CL`
RansomEXX	*!NEWS_FOR_STJ!.txt*: `ST→OP→ST→WR→CL`
REvil	*qoxaq-readme.txt*: `OP→ST→WR→CL`
Erebus	*_DECRYPT_FILE.html*: `OP→WR→CL→RN→OP→WR→CL` *_DECRYPT_FILE.txt*: `OP→WR→CL`
Darkside	*darkside_readme.txt*: *(1):* `ST` *(2):* `ST→OP→WR→CL` *(3):* `ST→OP→WR→CL→ST`

> **CRIBA's role.** *The aforementioned statistics were automatically computed by* CRIBA, *with the UNExt, DsetU and Transversals algorithms, and explored through its Generic Overview and Directory Transversal dashboards.*

B. Ransom notes

Table III shows the sequences of syscalls done for ransom notes by each ransomware family.

Observation 9. While the file name used for ransom notes changes across families, each sample reuses the same name for all written notes, most of which use the `.txt` extension. In AvosLocker, no extension is used, while Erebus creates an additional note with an `.html` extension.

Observation 10. Darkside creates 274 ransom notes, while RansomEXX, REvil and Erebus create more than 3500. Erebus creates more ransom notes with the `.html` extension (8430) than with the `.txt` one (4000).

Observation 11. While Darkside delegates the creation of ransom notes to two separate threads, the others use only one.

Observation 12. Most families exhibit a unique sequence of syscalls for ransom notes. Contrarily, Darkside does 3 different sequences, which is caused by having multiple threads creating these. Namely, each thread starts by performing a `stat` syscall to verify if the targeted directory already contains a note, writing a new one only if this is not the case.

Observation 13. All families perform `open`, `write` and `close` syscalls over each ransom note file. `Stat` is also significantly used (except for Erebus), but its amount and placement, in the sequence of syscalls issued per ransom note, varies (*i.e.*, AvosLocker and REvil only perform

TABLE IV: Syscall sequences issued per family over a small (*F31277.jsd.vswp*) and large (*F10573.bqt.vmdk*) dataset file.

Family	*Syscall sequence*	
	F31277.jsd.vswp (222 034 bytes)	*F10573.bqt.vmdk* (≈32 MiB)

AvosLocker	`OP→ST→LS→RD→`
	`LS→RD→LS→WR→CL→RN`
		`OP→ST→LS→RD→`
		`(LS→RD→LS→WR)x4→CL→RN`

RansomEXX	`OP→ST→LS→RD→WR→LS→RD→LS→WR→CL→RN`	

REvil	`OP→LS→ST→RD→LS→`
	`WR→LS→RD→WR→CL→RN`
		`OP→LS→ST→RD→LS→WR→`
		`(LS→RD→LS→WR)32→`
		`LS→RD→WR→CL→RN`

Erebus	`ST`	*original file*: `ST→RN`
		renamed file: `ST→OP→RD→LS→`
		`RD→WR→LS→WR→(LS→RD)x3→`
		`LS→WR→(RD→WR)x64→CL`

Darkside	`ST`	`ST→OP→(RD→WR)x32→`
		`CL→OP→LS→WR→CL→RN`

`stat` before opening the file, while RansomEXX performs it before and after opening the file).

Observation 14. The syscall sequence done for Erebus's `.html` ransom notes includes a rename (`RN`) operation. By further analyzing the arguments of the syscalls issued to these files, one observes that Erebus first creates and writes a file named `_DECRYPT_FILE.html`. Later, it renames the file to `index.html`, and creates another `_DECRYPT_FILE.html` file, in the same folder, with the same content.

> **Takeaways.** *The 5 ransomware families share similarities regarding the creation of ransom notes, such as reusing the same name for files placed across different directories and using almost the same set of syscalls. The observations also show distinct patterns, such as the number of ransom notes created by each family, the use of two distinct ransom notes by* Erebus*, and the different syscall sequences performed by* Darkside*.*

> **CRIBA's role.** *The previous findings were obtained through* CRIBA*'s UNext, FSysSeq, FnGrams, and Transversals algorithms, and by exploring their output with the File Name and Extensions, Syscall Sequences, File Ngrams and Directory Transversal dashboards.*

C. Dataset's Files Access and Encryption

Table IV shows syscall sequences done over a small (*F31277.jsd.vswp*) and large (*F10573.bqt.vmdk*) dataset file.

Observation 15. The number of unique sequences of syscalls, considering all dataset files, varies from 7 (REvil) to 28 (RansomEXX). These differ depending on specific file characteristics (*e.g.*, file size). For instance, RansomEXX uses the same sequence for the small and the large file examples, while the other families use different sequences. For AvosLocker and REvil, the difference is mostly on the amount of `lseek`, `read`, and `write` operations done to each file.

Observation 16. By inspecting the thread ID associated with each sequence, one can conclude that AvosLocker, REvil, and Erebus use 1 thread for accessing files, while Darkside uses 2 and RansomEXX uses 4. Further, except for Erebus and Darkside, the creation of ransom notes and the access to dataset files are delegated to distinct threads.

Observation 17. Some of the observed sequences only include the `stat` syscall. This suggests that some files are not being processed (*e.g.*, encrypted). In fact, through CRIBA, we can conclude that Erebus accesses all dataset's files but only processes 33.82% of these. Also, from the 30% of dataset's files accessed by Darkside, only 3.48% are being processed.

Observation 18. Erebus is the only sample accessing and encrypting files from all directories of the file system, which explains the high number of accessed files (*Observation 3*).

Observation 19. When inspecting the offset of `lseek` and `write` requests for other sequences, one can observe interesting patterns associated with the writing of encryption keys to infected files. Namely, RansomEXX starts by jumping to the end of the infected file (`ST→LS→RD`), writing the key (`WR`), and then jumping back to the beginning of the file to initiate the encryption process (`LS`). AvosLocker, REvil, and Darkside only write the key after encrypting the file's content. The latter reopens the file, jumps to its end, and then performs the key write operation (`OP→LS→WR→CL`). Erebus does not exhibit the previous file offset access patterns, as it writes the encryption key at the beginning of files before encrypting their content [23].

Observation 20. REvil and Erebus always read content from the `/dev/urandom` file before, or in between, consecutive accesses to each dataset file. The n-grams (`/dev/urandom`, `.../F3737.vva.vmem`, `/dev/urandom`, `.../F10573.bqt.vmdk`) from REvil and (`.../F10573.bqt.vmdk`, `/dev/urandom`, `.../F10573.bqt.vmdk`) from Erebus show these two patterns. Darkside also reads from the `/dev/urandom` file multiple times. These accesses are probably due to the generation of randomness for creating encryption keys.

Observation 21. Most of the ransomware families operate directly over the original dataset file and, after encrypting it, rename the file to add their own extension (note the `RN` operation at the end of most sequences). However, Erebus starts by first renaming the original file to a random name (*e.g.*, `F10573.bqt.vmdk` → `CA2065AE397D85C1.ecrypt`) and only then encrypts its content.

Observation 22. While most families add a constant file extension to encrypted files (*e.g.*, `.avoslinux` for AvosLocker, `.ecrypt` for Erebus), RansomEXX generates a different extension for each file, composed of a constant prefix (`.stj888-`) concatenated with a random suffix. Interestingly, some files have two distinct extensions (*e.g.*, the large file has the extensions: `.stj888-36acf3f1` and `.stj888-40aa97db`). By observing in more detail the syscalls issued for the large file by thread, as depicted in Figure 3, it is possible to see two threads simultaneously opening, reading, writing, and renaming the same file. This concurrent pattern can lead to data corruption and irrecoverable files.

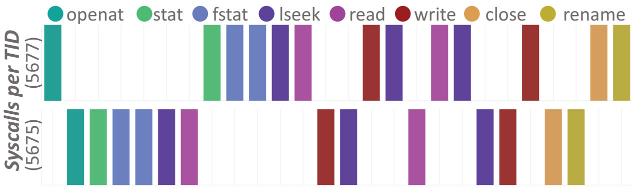


Fig. 3: Syscalls issued over time by RansomEXX's encryption threads to file F10573.bqt.vmdk.

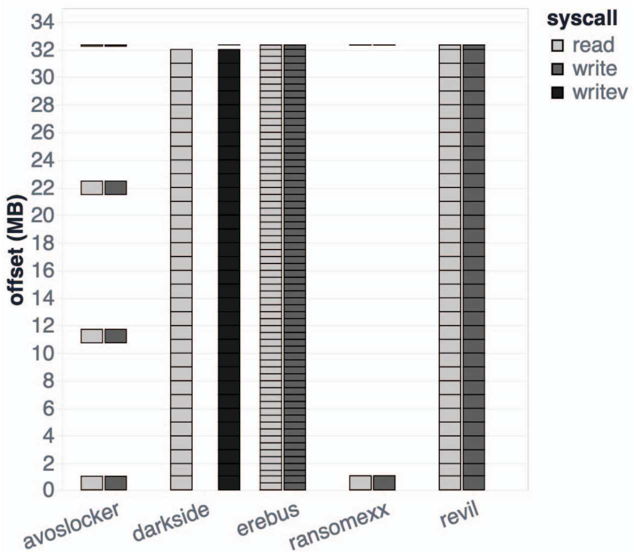


Fig. 4: File offsets accessed per family when reading and writing file F10573.bqt.vmdk.

> **Takeaways.** *The sequences of syscalls change according to the targeted files and the ransomware family. Different patterns are also observed regarding the timing and placement of encryption keys at infected files and for the extensions chosen by each family when renaming encrypted files. Interestingly,* REvil, Erebus, *and* Darkside *use* `/dev/urandom` *to generate randomness. In* RansomEXX, *two threads are concurrently encrypting the same file, a pattern that may lead to corrupted files.*

> **CRIBA's role.** *Results were obtained with the UNext, DsetU, FSysSeq, FnGram and Transversals algorithms, and observed with the Syscall Sequences, File Ngrams, File Offsets and Directory Transversal dashboards.*

D. Dataset's Files Selection and Evasion Techniques

Figure 4 shows the offsets accessed when reading and writing to the large file (*i.e.*, `F10573.bqt.vmdk`).

Observation 23. REvil and Erebus read and overwrite the full file's content (*i.e.*, offsets are fully accessed across the whole file ≈ 32.33MiB). Darkside encrypts most of the file's content but leaves the last (incomplete) block in plaintext. Also, REvil and Darkside use blocks of 1 MiB to read and write the file, while Erebus uses blocks of 512 KiB.

Observation 24. For every 10.78 MiB of content belonging to a given file, AvosLocker only encrypts 0.98 MiB. For the `F10573.bqt.vmdk` file, RansomEXX only encrypts the first 1 MiB block. However, CRIBA shows that the latter behavior changes across files and, for other files, this sample sparsely encrypts multiple 1 MiB blocks across the entire file.

Observation 25. *Observations 3 and 17* show that, except for REvil, all other families avoid encrypting the full dataset. When correlating the files accessed by Darkside with the dataset's file sizes and extensions, we observe that it only processes 4 types of extensions (*i.e.*, `.vmem`, `.vswp`, `.log` and `.vmdk`) and that only considers files with a size larger than 1MiB, amounting to 381 files.

Observation 26. As depicted in Figure 5, Darkside and RansomEXX have the highest bursts of CPU usage (94% and 86%, respectively), but for a short time, given the multiple threads doing data encryption. REvil exhibits a CPU usage of $approx$ 55%, for a larger time period, given its single thread encrypting the full dataset's content.

> **Takeaways.** *Only* REvil *accesses all dataset files and overwrites their full content. The other families target specific file extensions (e.g.,* Darkside*) and/or do not process the full content of files (e.g.,* AvosLocker*). These patterns enable faster execution and lower CPU usage and are used to deceive detection tools (see §II-D).*

> **CRIBA's role.** *DsetU and UNExt algorithms, along with the File Offsets, Directory Transversals, and Resource Usage dashboards were used for the previous observations.*

E. Families Similarity and Summary

Figure 6 highlights the similarity across families for the type of syscalls done and the file extensions and names accessed.

For the type of syscalls, Darkside is the most unique sample, sharing less than 22% of similarity with AvosLocker and REvil, and less than 55% with Erebus and RansomEXX. This is due to Darkside using more types of syscalls, including network calls (*Observations 5 and 7*).

When looking at accessed file extensions, Erebus is the sample with the highest deviation, being only 42.8% similar to REvil. This is explained by its distinctive behavior of encrypting files only after adding the *.ecrypt* extension (*Observation 21*), and by the number of accesses to `/dev/urandom`, which does not have an extension (*Observation 20*).

As for file names, the families are very dissimilar, with only Erebus, REvil, and Darkside sharing similarities (94.5% between the first two and up to 45.9% with the latter). The similarity between these 3 families results from their accesses to */dev/urandom* (*Observation 20*).

The results discussed in this section show that CRIBA provides comprehensive analysis and comparison of cryptographic ransomware samples. Also, it shows that ransomware

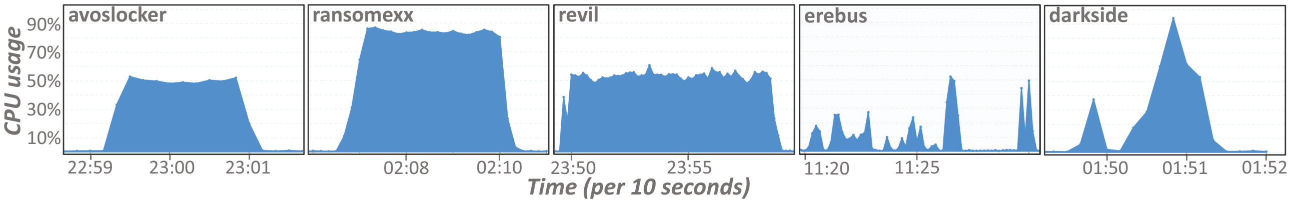
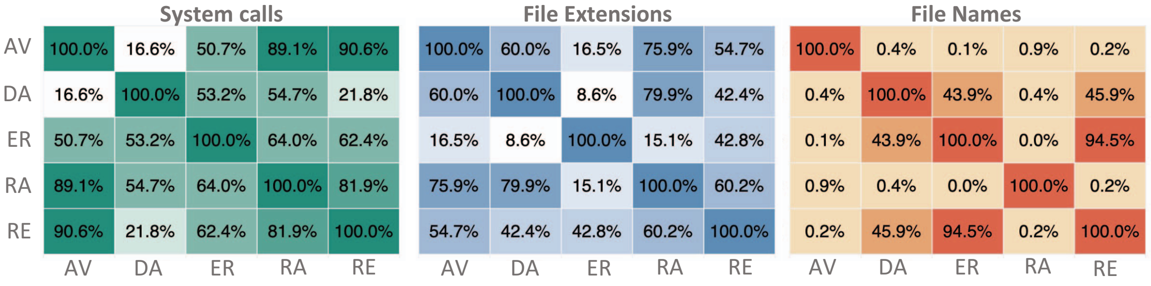


Fig. 5: CPU usage per ransomware family.



System calls

	AV	DA	ER	RA	RE
AV	100.0%	16.6%	50.7%	89.1%	90.6%
DA	16.6%	100.0%	53.2%	54.7%	21.8%
ER	50.7%	53.2%	100.0%	64.0%	62.4%
RA	89.1%	54.7%	64.0%	100.0%	81.9%
RE	90.6%	21.8%	62.4%	81.9%	100.0%

File Extensions

	AV	DA	ER	RA	RE
AV	100.0%	60.0%	16.5%	75.9%	54.7%
DA	60.0%	100.0%	8.6%	79.9%	42.4%
ER	16.5%	8.6%	100.0%	15.1%	42.8%
RA	75.9%	79.9%	15.1%	100.0%	60.2%
RE	54.7%	42.4%	42.8%	60.2%	100.0%

File Names

	AV	DA	ER	RA	RE
AV	100.0%	0.4%	0.1%	0.9%	0.2%
DA	0.4%	100.0%	43.9%	0.4%	45.9%
ER	0.1%	43.9%	100.0%	0.0%	94.5%
RA	0.9%	0.4%	0.0%	100.0%	0.2%
RE	0.2%	45.9%	94.5%	0.2%	100.0%

AV – avoslocker, DA – darkside, ER – erebus, RA – ransomexx, RE - revil

Fig. 6: Heatmaps comparing the families regarding the type of issued syscalls, and accessed file extensions and names.

analyses must consider different features to provide a clear understanding of the samples' intrinsic and complex behaviors.

VI. Related Work

Behavior analysis sandboxes provide a controlled environment for running malware samples and extracting their behavioral information [10], [11], [29]. These tools monitor the memory state, network traffic, and API calls done by samples and generate a report highlighting their main (and suspicious) activities. Raw logs are also outputted so that users can further analyze specific features not included in the report.

Leaving the parsing, analysis, and visualization of information contained in raw logs for users is a complex and time-consuming endeavor. Therefore, CRIBA can complement these tools with the mechanisms for automating the aforementioned tasks. For instance, security analysts can use tools like Cuckoo Sandbox [10] to perform a first analysis of multiple malware and benign applications. Based on their reports, they can select the samples that need a more in-depth analysis and use our tool to ease such a process.

Ransomware detection tools also rely on dynamic analysis for understanding the key features that identify ransomware samples exhibiting malicious activity [6]–[8], [17], [30]–[33]. These tools typically resort to the aforementioned analysis sandboxes to collect information from the samples' execution and then use feature selection techniques to extract the most relevant features for malware detection [34], [35].

As discussed in VizMal [36], since the purpose of detection tools is to classify applications as malign or benign, they do not provide further information for exploring and understanding the internal behavior of ransomware samples. VizMal introduces a new visualization to highlight potential malicious behavior at specific portions of the execution traces of Android ransomware samples. Nonetheless, VizMal is limited to this single visualization, and therefore, it cannot provide comprehensive information about ransomware's I/O behavior.

CRIBA is not a tool for ransomware detection, instead, its main goal is to provide an integrated tracing and analysis pipeline optimized for collecting, exploring, and visualizing a vast amount of I/O information about ransomware samples. With this information and the aid of custom correlation algorithms and visualizations, CRIBA automates the observation of interesting I/O patterns for ransom note creation (§V-B), data encryption (§V-C) and evasion techniques (§V-D) used by ransomware samples. Also, it enables the comparison of different samples from either the same or different families while pinpointing their main similarities and differences (§V-E).

VII. Conclusion

We present CRIBA, a tool for simplifying and automating the exploration, analysis, and comparison of I/O patterns for Linux cryptographic ransomware. CRIBA supports non-intrusive and comprehensive collection of I/O information from ransomware samples and combines it with an integrated analysis and visualization pipeline. The latter is enhanced with 6 custom correlation algorithms and different pre-defined dashboards. As shown in our experimental study, these features are key to: *i)* automate the analysis of ransomware families; *ii)* understand complex and intrinsic behavior from each sample; *iii)* and pinpoint common and distinct traits across families.

Acknowledgment

This work was financed by the FCT - Portuguese Foundation for Science and Technology, through Ph.D. grant DFA/BD/5881/2020 (*Tânia Esteves*), and realized within the scope of the project UIDP/50014/2020 (*João Paulo).*

References

[1] R. Brewer, “Ransomware attacks: detection, prevention and cure,” *Network Security*, vol. 2016, no. 9, pp. 5–9, 2016.

[2] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, “A survey on ransomware: Evolution, taxonomy, and defense solutions,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022.

[3] “What We Know About the DarkSide Ransomware and the US Pipeline Attack,” 2023. [Online]. Available: https://www.trendmicro.com/en_us/research/21/e/what-we-know-about-darkside-ransomware-and-the-us-pipeline-attac.html

[4] “Ransomware Spotlight: REvil,” 2023. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-revil

[5] “Ransomware Spotlight: RansomEXX,” 2023. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-ransomexx

[6] D. Hitaj, G. Pagnotta, F. De Gaspari, L. De Carli, and L. V. Mancini, “Minerva: A File-Based Ransomware Detector,” *arXiv preprint*, 2023.

[7] S. K. Shaukat and V. J. Ribeiro, “RansomWall: A Layered Defense System against Cryptographic Ransomware Attacks using Machine Learning,” in *10th international conference on communication systems & networks (COMSNETS)*, 2018, pp. 356–363.

[8] C. J.-W. Chew, V. Kumar, P. Patros, and R. Malik, “ESCAPADE: Encryption-Type-Ransomware: System Call Based Pattern Detection,” in *14th International Conference on Network and System Security (NSS)*, 2020, pp. 388–407.

[9] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, “Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions,” *Computers & Security*, vol. 74, pp. 144–166, 2018.

[10] C. Foundation, “Cuckoo: Automated Malware Analysis,” 2023. [Online]. Available: https://cuckoosandbox.org

[11] “Limon - Sandbox for Analyzing Linux Malwares,” 2023. [Online]. Available: https://github.com/monnappa22/Limon

[12] P. O’Kane, S. Sezer, and D. Carlin, “Evolution of ransomware,” *IET Networks*, vol. 7, no. 5, pp. 321–327, 2018.

[13] E. Berrueta, D. Morato, E. Magaña, and M. Izal, “A Survey on Detection Techniques for Cryptographic Ransomware,” *IEEE Access*, vol. 7, pp. 144 925–144 944, 2019.

[14] T. Esteves, R. Macedo, R. Oliveira, and J. Paulo, “Diagnosing applications’ I/O behavior through system call observability,” in *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2023.

[15] S. Halim and F. Halim, *Competitive Programming 3: The New Lower Bound of Programming Contests*, 3rd ed. Lulu.com, 2013.

[16] Q. Chen, S. R. Islam, H. Haswell, and R. A. Bridges, “Automated Ransomware Behavior Analysis: Pattern Extraction and Early Detection,” in *2nd International Conference on Science of Cyber Security (SciSec)*, 2019, pp. 199–214.

[17] Q. Chen and R. A. Bridges, “Automated Behavioral Analysis of Malware: A Case Study of WannaCry Ransomware,” in *16th IEEE International Conference on machine learning and applications (ICMLA)*, 2017, pp. 454–460.

[18] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli, and A. K. Sangaiah, “Classification of ransomware families with machine learning based on N-gram of opcode,” *Future Generation Computer Systems*, vol. 90, pp. 211–221, 2019.

[19] E. B.V., “Elasticsearch: The heart of the free and open Elastic Stack,” 2023. [Online]. Available: https://www.elastic.co/elasticsearch/

[20] ——, “Kibana: Your window into the Elastic Stack,” 2023. [Online]. Available: https://www.elastic.co/kibana/

[21] ——, “Metricbeat: Lightweight shipper for metrics,” 2023. [Online]. Available: https://www.elastic.co/beats/metricbeat

[22] “Vega-Lite – A Grammar of Interactive Graphics,” 2023. [Online]. Available: https://vega.github.io/vega-lite/

[23] “Erebus Resurfaces as Linux Ransomware,” 2023. [Online]. Available: https://www.trendmicro.com/en_se/research/17/f/erebus-resurfaces-as-linux-ransomware.html

[24] “Ransomware Spotlight: AvosLocker,” 2023. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-avoslocker

[25] “DarkSide Ransomware as a Service (RaaS),” 2023. [Online]. Available: https://www.state.gov/darkside-ransomware-as-a-service-raas/

[26] D. Morato, E. Berrueta, E. Magaña, and M. Izal, “Ransomware early detection by the analysis of file sharing traffic,” *Journal of Network and Computer Applications*, vol. 124, pp. 14–32, 2018.

[27] E. Berrueta, D. Morato, E. Magaña, and M. Izal, “Open Repository for the Evaluation of Ransomware Detection Tools,” *IEEE Access*, vol. 8, pp. 65 658–65 669, 2020.

[28] N. Agrawal, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “Generating Realistic Impressions for File-System Benchmarking,” *ACM Transactions on Storage (TOS)*, vol. 5, no. 4, pp. 1–30, 2009.

[29] B. Baskin, “Noriben Malware Analysis Sandbox,” 2023. [Online]. Available: https://github.com/Rurik/Noriben

[30] A. Kharraz, S. Arshad, C. Mulliner, W. K. Robertson, and E. Kirda, “UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware,” in *25th USENIX Security Symposium*, 2016.

[31] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, “CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data,” in *36th international conference on distributed computing systems (ICDCS)*, 2016, pp. 303–312.

[32] N. Andronio, S. Zanero, and F. Maggi, “HelDroid: Dissecting and detecting mobile ransomware,” in *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, vol. 9404, 2015, pp. 382–404.

[33] K. Tam, A. Fattori, S. Khan, and L. Cavallaro, “CopperDroid: Automatic Reconstruction of Android Malware Behaviors,” in *NDSS Symposium*, 2015, pp. 1–15.

[34] P. M. Anand, P. S. Charan, and S. K. Shukla, “A Comprehensive API Call Analysis for Detecting Windows-Based Ransomware,” in *IEEE International Conference on Cyber Security and Resilience (CSR)*, 2022, pp. 337–344.

[35] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, “Cutting the gordian knot: A look under the hood of ransomware attacks,” in *12th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2015, pp. 3–24.

[36] A. De Lorenzo, F. Martinelli, E. Medvet, F. Mercaldo, and A. Santone, “Visualizing the outcome of dynamic analysis of Android malware with VizMal,” *Journal of Information Security and Applications*, vol. 50, p. 102423, 2020.

Appendix

A. Supported system calls

TABLE V: List of syscalls supported by CRIBA and corresponding tags, used by the *FSysSeq* algorithm (§III-D).

Tag	Syscall
AC	accept, accept4
BD	bind
CL	close
CN	connect
CR	creat
FS	fsync, fdatasync
GS	getsockopt
GX	getxattr, lgetxattr, fgetxattr
LS	lseek
LT	listen
LX	listxattr, llistxattr, flistxattr
MK	mknod, mknodat
OP	open, openat
RC	recvfrom, recvmsg
RD	read, pread64, readv
RH	readahead
RL	readlink, readlinkat
RN	rename, renameat, renameat2
RX	removexattr, lremovexattr, fremovexattr
SD	sendto, sendmsg
SK	socket, socketpair
SS	setsockopt
ST	stat, lstat, fstat, fstatfs, fstatat
SX	setxattr, lsetxattr, fsetxattr
TR	truncate, ftruncate
UN	unlink, unlinkat
WR	write, pwrite64, writev

B. Dataset statistics

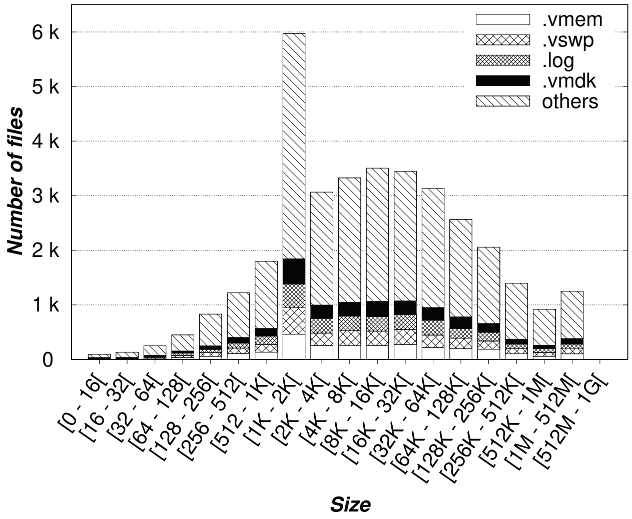


Fig. 7: Distribution of file sizes and extensions for the dataset used in §V (*others* aggregates extensions other than *.vmem*, *.vswp*, *.log*, and *.vdmk*).

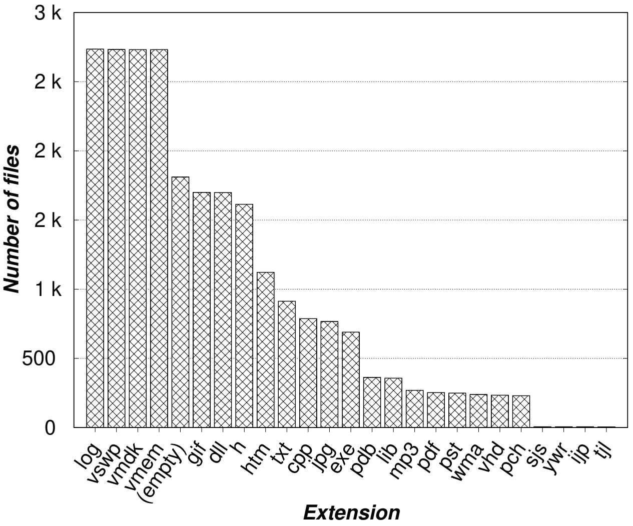


Fig. 8: Number of files with the top 25 file extensions from the dataset used in §V.

C. Ransomware samples

TABLE VI: SHA256 hashes and execution commands for the 5 ransomware samples analyzed in §V.

Family	*SHA256*	*Command*
AvosLocker	d7112a1e1c68c366 c05bbede9dbe782b b434231f84e5a72a 724cc8345d8d9d13	./avos.elf 1 /app/files
RansomEXX	08113ca015468d6c 29af4e4e4754c003 dacc194ce4a254e1 5f38060854f18867	./ransomexx.elf --threads 1 --path /app/files
REvil	3d375d0ead2b6316 8de86ca2649360d9 dcff75b3e0ffa2cf 1e50816ec92b3b7d	./revil.elf --path /app/files --threads 1
Erebus	0b7996bca486575b e15e68dba7cbd802 b1e5f90436ba23f8 02da66292c8a055f	./erebus.elf
Darkside	c93e6237abf041bc 2530ccb510dd016e f1cc6847d43bf023 351dce2a96fdc33b	./darkside.elf --path /app/files

D. CRIBA*'s visualizations*

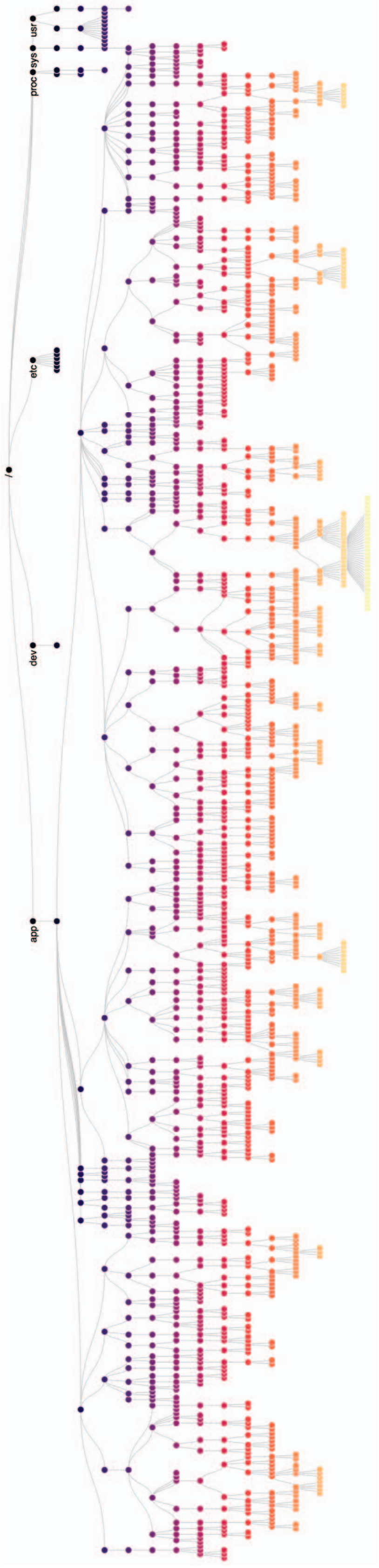


Fig. 9: Visual tree representation for the files and directories accessed by DARKSIDE.