On the Inadequacy of Open-Source Application Logs for Digital Forensics

Afiqah M. Azahari[1], Davide Balzarotti[a]

[a] *Eurecom, Sophia Antipolis, France*

Abstract

This study explores the challenges with utilizing application logs for incident response or forensic analysis. Application logs have the potential to significantly enhance security analysis as sometimes they provide information regarding user actions, error messages, and performance metrics of the application. Although these logs can offer vital information about user activities, errors, and application performance, their use for security needs better understanding. We looked at the current logging implementation of 60 open-source applications. We checked the logs to see if they could help with five key security tasks: making timelines, linking events, separating different actions, spotting misuse, and detecting attacks. By examining source code, extracting log statements, and evaluating them for security relevance, we found many logs lacked essential elements. Specifically, 29 applications omitted timestamps, crucial for identifying the timing of actions. Furthermore, logs frequently missed unique identifiers (UIDs) for event correlation, with 23 not noting UIDs for new activities. Inconsistent logging of user activities and an absence of logs detailing successful attacks indicate current application logs need significant enhancements to be effective for security checks. The findings of our research suggest that current application logs are inadequately equipped for in-depth security analysis. Enhancements are imperative for their optimal utility. This investigation underscores the inherent challenges in leveraging logs for security and emphasizes the pressing need for refining logging methodologies.

Keywords: Logs, application, forensic, security

Email addresses: `afiqah.azahari@eurecom.fr` (Afiqah M. Azahari), `davide.balzarotti@eurecom.fr` (Davide Balzarotti)

1. Introduction

In the evolving digital ecosystem, the significance of application logs for ensuring robust computer systems cannot be overstated. In particular, while their main purpose remains troubleshooting, their use for security is rapidly gaining importance. For instance, they can play a key role in flagging unusual or unauthorized activities and they can provide valuable insights into the functioning of an application, allowing administrators to detect unauthorized access attempts or system component errors. Consistent examination of application logs also enables the timely detection and mitigation of security threats. Extensive research has been conducted by the software engineering community to improve application logs, focusing on performance monitoring, software failure pinpointing, and on enhancing the logging process during software development [1, 2, 3, 4]. Researchers have also analyzed log patterns to propose advancements and automation in logging methodologies, aiming to enhance the efficiency and effectiveness of logging practices in different application types.

From a security perspective, existing literature has mainly focused on proposing automated suggestions to refine application logging techniques [5, 6, 7, 6]. However, these studies provide recommendations suitable for only a narrow range of attacks, and often advocate techniques that are applicable to a specific subset of application types. Moreover, while the broader utility of application logs in areas like monitoring, troubleshooting, and evaluating security breaches were acknowledged, there is a lack of comprehensive research on their use for detailed forensic analysis in various user applications. Our research aims to address this gap by conducting a thorough examination of the challenges and limitations associated with utilizing application logs for security purposes. We highlight specific areas that require improvement to enhance their effectiveness in security-related applications and discuss inadequacies or inconsistencies that could impede their efficacy in security-related use.

Our first objective was to propose a taxonomy of the possible uses of application logs for forensic purposes, and to then study which information logs need to contain to perform such actions. We then embarked on a comprehensive study of 60 different open source applications. We studied their source code, extracted relevant log lines, and assessed their current implementation according to our requirements. Our primary goal was two-fold: to identify potential forensic tasks that can be performed using these application logs and to propose guidelines emphasizing the essential parameters these logs should capture.

The results of our experiment helped us uncover significant insights. In particular, we found that

the majority of logs from the studied applications were found to be inadequate when evaluated for incident response or forensic analysis. This highlights the need for more effective logging practices for security and forensic purposes in user applications. By addressing these findings, our research aims to contribute to the development of enhanced logging systems, ultimately improving the efficacy of security measures in various domains.

In summary, this paper makes the following contributions:

- We characterized five common forensic tasks, including timeline activity, event correlation, execution partitioning, misuse detection, and attack detection, and emphasized their importance in the context of security and application logs.
- We identified parameters that application logs should embody, such as timestamps, unique identifiers for event correlation and execution partitioning, documentation of user actions, and warning related to the processing of unusual or malformed data. These parameters are crucial for incident response and forensic analysis.
- We conducted a detailed manual assessment on 60 open-source applications to understand current practices of application logs for forensic purposes and to identify potential challenges.
- Based on our analysis, we describe existing challenges in using application logs for security purposes, including insufficient timestamp data in 29 applications, a high percentage of text-only log events in more than half of the applications, inconsistent logging of specific user actions, and inadequate logging of exceptions in 35 applications.

2. Background

The term *log* originally referred to the process of keeping a record of the progress of a ship, an operation that was performed by using a wooden chip log attached to a knotted rope [8]. In the context of computing and information technology, the term is used to describe the recording of events that capture the usage, the activity, and the operation of a computer system.

2.1. Terminology

Logs are routinely used to record information about the runtime operations of operating systems, servers, applications, and network devices. This data is usually broken down in a sequence

of individual *entries*, each reporting information on a particular event that occurred within the source that originates the log. For example, operating system logs capture a range of events from startup messages and system modifications to unexpected shutdowns. In contrast, application logs provide a comprehensive overview of activities, which encompass a broader scope of actions and user interactions within the application. These logs might include detailed records such as authorization processes, descriptions of actions taken, identifications of users involved, explanations for any failures, and records of activities between the application and its users, as well as insights into the internal dynamics of the application.

Logs can be stored in various formats and locations, such as files, databases, or cloud-based storage systems, and serve as a primary source of information that is essential for detecting and troubleshooting problems but also for verifying the operational performance and efficiency of a system.

Logs also play a crucial role for security, both as a way to **detect** attacks or compromise and as a way to **investigate** them in a post-mortem scenario.

2.2. Standards and Guidelines

Various standards and guidelines exist to guide the developers and organizations to design and streamline their logging process. For instance, ISO 27002 provides essential logging requirements, such as the fact that audit logs should be generated and stored for a certain period of time to assist future analysis. At the same time, ISO provides very few details on how such requirements needs to be implemented.

The Control Objectives for Information and Related Technologies (COBIT) framework offers additional information about logging management, including logging configuration, changes, and backup processes. Nevertheless, COBIT provides little details regarding the requirement of logs for security purposes. Another industry standard that specifies the requirement to log is PCI. In fact, PCI v3.2.1 provides much more detailed logging guidance that any relevant applications could apply. These include logs' details and properties, the type of events to logs, the importance of time synchronization, important security requirement, logs locations, log retention, and the need for a routine review [9]. While the PCI standard provides a complex methodology, it is specifically designed for software applications, which require different logging provisions and practices.

Additionally, many technical blog posts provide logging recommendations [10, 11, 12, 13, 14,

15, 16, 17]. Industry players, support engineers, the development community, and security experts often discuss logging and share their opinions and perspectives on proper logging techniques. These posts provide tips, including what to log, what 'not' to log, which verbosity level and logging format to use, and which log retention approach to adopt.

One major limitation is that the vast majority of log standards and guidelines focus their requirements and recommendations on logs for debugging rather than for security. In 2010 Chuvakin and Peterson [18] compared the two classes (debugging vs security) and discussed how the second should be intended specifically for security and audit personnel. Security-relevant logs should always be available and contain messages reporting information about attacks, activities, and faults. The authors also defined what to log and the retention of log files.

Moreover, Brouwer and Mertens [19] listed five categories of logging requirement that are relevant for forensic investigation. These requirements include logs retention, correlation, content, integrity and consideration. The authors mention that logs for security should contain information on *"who did what, where, when, how, and the results"*. However, these suggestions and recommendations were based on the authors personal experience, without any justification or experiment to support the discussions.

Finally, it is important to mention that the Cyber-Investigation Analysis Standard Expression (CASE) introduces a significant effort to standardize the representation of forensic data, thus simplifying the investigation and analysis process through enhanced provenance and traceability [20]. CASE considerably improves the existing standards and guidelines for logging in security and forensic contexts. By breaking down data silos and enhancing tool interoperability, CASE supports the automated normalization, combination, and correlation of data from diverse sources. Its community-driven API eases integration into various tools, allowing developers to tailor CASE to fit their specific data structures and formats [21, 22]. This flexibility in mapping and converting data to comply with the CASE standard makes it a valuable asset in advancing digital forensic investigations and facilitating collaboration among cyber-investigators. However, integrating CASE without properly understanding the essential information required for forensic logging may lead to inadequate evidence collection. Additionally, the use of CASE to merge multiple data sources, especially those lacking comprehensive data, might backfire and actually impede effective evidence extraction. Thus, while CASE presents a comprehensive framework for integrating diverse data sources, the accuracy and completeness of the input data remain critical for effective forensic evidence gathering

and analysis.

2.3. Related Work

Previous studies on logging in software applications tackled the problems of where-to-log, what-to-log, and how-to-log. Multiple studies ventured into the problem by trying to understand how developers log in terms of their logs level, log descriptions, logs history, and logs variable.

Several papers [23, 24] performed studies to understand the placement of logs in existing applications, with the goal of proposing automated logging placement techniques that should help programmers during the development phase. For instance, Cinque et al. [25] performed a preliminary study on the log placement on eight software platforms to understand the limitation of current logging mechanisms. Yao et al. [26] suggested a framework to improve the location of logging for performance monitoring. Zhu et al. [27] proposed a framework to help developers make informed logging decisions on where to log, therefore reducing their logging efforts. Ding et al. [28] suggested a low-cost logging framework that can reduce overhead while maintaining a high effectiveness.

Multiple studies have been done to improve the implementation of log levels, log descriptions, and log variables. Li et al. [29] analyzed the development history of four open-source code projects, then created a model to automatically suggest the most appropriate level for each newly-added logging statement. He et al. [30] studied the purpose of logging descriptions of 10 Java projects and seven C# projects and created a tool that generated automatic logs description. Moreover, as logging statements usually prints one or more variables to record vital system status, Liu et al. [31] proposed an approach to recommend logging variables. Researchers [32] [33] have also studied the history and evolution of logging statements to improve current implementations by automatically detecting new problematic logging codes and anti-patterns in logging, and then report the result to the developers for further improvement. Replication studies have been made by Chen et al. [34] to assess whether the finding from [32] would apply to Java projects part of the Apache Software Foundations. The authors found that all assessed Java projects contain logging code that is actively maintained. Nevertheless, in contrast with the original study, they also found that any bug reports containing log messages take longer to resolve compared to bug reports without. Kabinna et al. [35] also studied the history of logging code. In this case, instead of looking for problematic logging statements, the authors examined whether the introduced or long-lived logging statements are likely to change.

User studies also exist in this area. For instance, Pecchia et al. [36] tried to understand how developer logs, why they log and how event logging is implemented in the industry. To gain broad insight from programming practices, logging objectives and issues impacting log analysis, they combined source code analysis on 2.3 million log entries and obtained direct feedback from the development team. In conclusion, they find that the process of logging strongly relies on human expertise.

Among the existing studies, some have also focused on logging for security. King et al. [37] performed a study to evaluate logging practices for security incidents detection on an open-source health care software. The authors performed a black box test and evaluated the collected logs based on the Standard Specification for Audit and Disclosure Logs for Use in Health Information Systems. They also developed a forensic-ability metric to measure the coverage of mandatory log events for user activity logs [5]. To do that, the authors manually identified all pairs of verbs and objects that act upon user activity of the studied software to develop the mandatory log events. While this was an interesting experiment, the finding were very specific for this domain and this particular application, and the approach did not suggest how logging should be done to capture user activity. In 2019, Rivera-Ortiz and Pasquale presented a preliminary idea to automate forensic-ready software system development. They developed a framework that allows security engineers to replay potential security incidents [7] and then provided a semi-automated approach to automatically inject logging statements based on the replay framework [6]. However, the solution was limited to a specific type of system (JAVA apps built on top of a MySQL database).

In conclusion, despite the numerous studies that have been conducted on the topic of logging, to the best of our knowledge no research has yet been conducted to specifically investigate the state of security- and forensic-relevant logs across many types of user-space applications.

3. Forensic Use of Application Logs

Application logs can provide a broad range of information that can be used as part of an incident response or forensic investigations. However, both the number and the type of tasks that an analyst can perform depend on the available data.

In this section we group common forensic analysis tasks in five main categories:

1. Activity Timelines.

Timelines play a very important role in digital forensics, as they allow an investigator to easily aggregate and navigate the sequence of events that took place on a computer system. For instance, a timeline can be used to identify the cause of an incident by locating co-occurring events (such as an external drive plugged in just before a document was accessed) or by identifying patterns of sequential behaviors. The ability to restrict the analysis to particular time ranges, to pinpoint relevant entries, and to put them in context with respect to all other pieces of evidence collected in the target system makes timelines an invaluable tool.

In order to be integrated into a timeline, log entries must include a well-identified timestamp. Their absence make it difficult or impossible to synchronize the logs with other data sources and correlate events.

In addition, the application needs to log relevant events that capture the user behavior (these events differ from one type of application to another, and we will discuss them in more details in Section 4).

2. Events Correlation.

   The ability to correlate events from different sources is paramount in a forensic investigation. For instance, a broad range of information can be extracted from emails, server logs, network appliances, antivirus products, and operating systems. All these sources can be useful in isolation, but the ability to 'connect the dots' by linking together events across the entire target system, or across different systems, can help an investigator to better understand what is relevant to the case and what parts of the system have been compromised. For instance, a succesful password bruteforce against a dormant employee account can be difficult to investigate if the application does not log the username associated to each attempt, or the network address from which the connections originated.

   The representation of unique characteristics in log message plays a crucial role in event correlation. In particular, the presence of unique identifiers (such as URLs, user names and IDs, network addresses, domain names, and complete file and directory paths) helps to reconstruct complex timelines of events and to transform raw logs into coherent event sequences, such as workflows. For instance, a recent study by Li et al. [3] leverages commonly available IDs within logs to tackle challenges associated with intermixed event sequences.

Moreover, unique events in logs enrich the analysis by tracing activity sequences within the application, providing additional context. Each unique event acts as a distinct marker or footprint, allowing the tracing of specific actions or occurrences.

The unique values provided by dynamic variables in logs aid in differentiating between normal and anomalous system behavior. Developers have observed that various categories of these dynamic variables capture crucial information, enhancing log analysis. Thus, these unique identifiers in logs can be harnessed to correlate events and identify system components that may require in-depth scrutiny.

3. Execution Partitioning.

   Audit logs are among the most precise and fine-grained source of information available to an investigator. When enabled, in combination with execution and application logs, they allow tracking of each system event's complex dependencies using *data provenance graphs* [38]. However, the presence of long-running processes often undermine the effectiveness of the analysis by introducing the provenance graph a large numbers of irrelevant or erroneous causal dependencies. For instance, a new file created by a browser would depend on all URLs that were ever opened by the same process in the past. This dependency explosion is a well-known problem in digital forensics, and over the years researchers have proposed several mitigations based on the idea of *execution unit partitioning* [39, 40, 41]. In a nutshell, partitioning consists in breaking down audit log events in small segments, based on the presence of particular application events. For instance, whenever an application closes a document and opens a new one, causal dependencies from the content of the former document should not be propagated to the latter.

   To support partitioning, a user application needs to generate a recognizable log entry in correspondence to a unit boundary (e.g., when serving a new user requests or processing a new document). This idea was recently described by Yu et al. [42] who in 2021 analyzed 32 Linux applications and found that 1) the vast majority were long-running and 2) that all of them had log entries that could be used for the purpose of execution partitioning.

4. Misuses detection.

   With this term we refer to the ability of an analyst to use application logs to identify unau-

thorized behaviors, typically associated to insider threats or to attackers that already gained access to the system. For instance, a legitimate user can try to misuse an application to perform lateral movements inside the network, to impersonate another user, or to gain access to sensitive information.

In order to support misuse detection an application needs to log any operation that affect the state of the system (such as change of privileges or service configurations). In addition logs should report operations on sensitive data as well as any action which failed – or which was denied – based on insufficient user permissions. A typical example is provided by the `sudo` setuid application, which logs any failed attempt to execute privileged commands. This information can serve as a way to support future investigations, but also as an effective way to deter users against poking around and trying to guess passwords.

5. Attack detection.

   After an attack, it is crucial for an investigator to be able to retrieve the attack vector, the source of the attack, and all actions that the attacker performed on the system. Security logs from both network and host intrusion detection systems are designed specifically for this purpose but they often lack the necessary visibility into each application state. For instance, they may be able to detect a malicious file created in the `/tmp/` directory but they might fail to provide information on which process created the file and, more importantly, which malicious input caused the process to misbehave.

   In the lack of audit logs, application logs could help to cover this blind spot by providing information about any unusual, suspicious, or malformed piece of data the application encounters. For instance, exceptions due to unexpected behaviors or triggered by malformed data and input that fails validation routines should be reported in the logs, together with enough information to identify the source of the data and/or its correspondent user request.

The aforementioned five categories cover many different uses of application logs for forensics and incident response. Each category introduces specific requirements (e.g., the presence of timestamps and unique object identifiers) and guidelines for developers (e.g., the need to log exceptions and malformed or invalid inputs).

An orthogonal dimension, which crosses all possible use cases, is related to the availability and integrity of log files. In practice, application logs are often written to multiple locations, including remote servers, consoles, files, and databases. Some applications print relevant error information only to the standard error, making them difficult to collect and store for later use. Log availability also need to consider other log properties such as log rotation, log archival, log retention period, and security properties such as access privileges or the use of append-only storage. Centralized log daemons simplify this procedure but they are typically adopted only to manage system services and servers, leaving other userspace applications to their own custom log management.

Creating a comprehensive list of user actions that an application should log to support security and forensic investigations is very difficult. Different applications serve different purposes, thus with different requirements, features, and development methodologies. For instance, Office Suites differ from a communication chats or file sharing applications.

To this extent, application developers choose different log frameworks, formats, and default configurations. In general, logs are created for the purpose of debugging and it is still unclear to which extent they can also satisfy the requirements of the five categories described above. We will try to answer this question in the next section.

4. Methodology

This section presents the list of applications we selected for our study and the methods we used to evaluate the usefulness of their logs for common forensic analysis tasks.

To conduct our study we selected 60 open-source applications across various categories, including file managers, chat and communication tools, office software, file sharing applications, screen capture programs, antivirus software, text editors, remote desktop applications, window managers, and application launchers. Our goal was to capture a wide range of applications, which can be used to process untrusted data or to interact with a system in a way that can be interesting to capture for a forensic investigation. We chose to focus on open-source applications as they provide access to the source code, thus enabling a thorough examination of the nature and content of their individual log statements.

Table 1: List of analyzed applications

Category	Application	Description
File Managers	Nautilus	Default file manager of the GNOME desktop
	Disks	Default storage management of the GNOME desktop
	Dolphin	KDE's file manager
	nnn	Terminal file manager
	recoll	Full-text search tool for Unix/Linux
	Konqueror	File management and preview based on web browsing
	Thunar	File manager for Linux and other Unix-like systems
Chat and Communication	IceChat (Windows)	IRC Client
	BeeBEEP	Peer-to-peer office messenger
	Pidgin	Multi-platform instant messaging client
	signal-desktop	A private messenger application
	Mattermost-Desktop	Platform for secure collaboration
	Konversation	User-friendly and fully-featured IRC client
	jitsi	VoIP, video conferencing, and instant messaging
	wire-desktop	Encrypted communication and collaboration app
	session-desktop	Onion routing-based messenger
	HexChat	IRC client
	Terminal-Irssi	Terminal-based IRC client
	tox	Peer-to-peer instant messaging and video-calling app
Office Tools	Calligra	Office and graphic art suite by KDE
	Xournal++	Handwriting note-taking software with PDF annotation support
	SumatraPDF	PDF Viewer
	zael	Offline documentation browser
	Okular	KDE document viewer
	xpdf	PDF viewer and toolkit
	Apache Open Office	Office productivity software suite
	Scribus	Libre Desktop Publishing
	Document Viewer	PDF Viewer for GNOME
Torrent and File Sharing Platforms	qBittorrent	BitTorrent client
	Deluge	Cross-platform BitTorrent client
	transmissiongtk	Transmission BitTorrent client repository
	frostwire	BitTorrent client
	warpinator	File sharing across the LAN
Screen Capture Tools	ShareX	Capture or record any area of the screen
	OBS-Studio	Live streaming and screen recording
	Greenshot (Windows)	Screenshot manager
Security and Antivirus	Seahorse	Password and encryption key manager for GNOME
	hashcat	Password recovery utility
	Clam AntiVirus (Linux)	Open-source antivirus
Text Editors	Zettlr	Markdown editor
	gnome-text-editor	Default editor of GNOME
	Vim	Highly configurable text editor
	jrnl	Simple journal application for the command line
Utilities	Console	GNOMEs default terminal emulator
	Cheese	GNOME's webcam application
	Guvcview	Webcam application
	XDM	Download manager
Media Players	VLC Media Player	Video player
	musikcube	Terminal-based music player
	Kodi	Media player and entertainment hub
Window Managers	IceWM	Open window manager
	tmux	Terminal multiplexer
	bspwm	Tiling window manager
	Qtile	Hackable tiling window manager
Remote Desktop Tools	Remotely	A remote control and remote scripting solution
	xrdp	Open-source RDP server
App Launchers	albert	A fast and flexible keyboard launcher
	launchy	Linux launcher
	ulauncher	Linux launcher
	Wox	Launcher for Windows

Table 1 presents an overview of our dataset. The applications we considered are developed in various programming languages, including Java, Python, C, C++, C#, and Typescript. Although some applications may contain components written in multiple languages (e.g., in the case of plugins or extensions), our analysis focused on the primary language used for the development of the core application.

We downloaded the latest source code of each applications and used the SLOCCOUNT tool[1] to count the lines of code (SLOC). We then employed a number of custom regular expressions to identify and count the lines of log code (LOLC). Although this approach is similar to the one adopted by previous studies (such as [23], [34], [32] and [33]), we payed particular attention to improve the accuracy of the results by manually curating and customizing the process based on each application's log framework or log method. This resulted in a set of custom regular expression specifically tailored to each application, which allowed us to collect all messages and properly recognize multi-line log statements.

We identified a number of features and parameters related to each log statement. Using this collected information, we verified whether the application can support the five forensic tasks identified in Section 3.

Timestamps

To support the process of timeline development and visual representation of the sequence of events, it is crucial for an application to provide log entries containing timestamp data.

Log2timeline is a very popular tool that significantly improves forensic analysis by allowing the creation of comprehensive timelines[2]. It is the de facto tool for parsing a variety of log files from different data sources, thereby creating detailed timelines that present a chronological sequence of events within the system. However, application logs lacking timestamp data become less useful for forensic analysis, especially when using tools like Log2timeline. In such scenarios, Log2timeline treats these logs as basic 'raw data' with limited analytical potential. This limitation impacts the ability to establish events in a chronological order, detect anomalies and patterns, and assess the duration and frequency of events.

Timestamps can be introduced explicitly by the developer into the logged message, or they can

[1] https://dwheeler.com/sloccount/

[2] https://github.com/log2timeline/plaso

be implicitly added (and often configured) by the logging framework adopted by the application.

To determine if timestamp data is present in the logs of the studied applications, we followed a series of steps. First, we identified the programming language and framework used in the application's source code because logging implementation can vary based on these factors. Second, we analyzed the logging configuration files or code to ascertain which logging library or tools were employed, thereby gaining insight into the available logging syntax and features. Third, we searched for logging statements in the source code and inspected them for the presence of timestamp-related parameters or functions. For instance, in Python's logging library, the format specifier can indicate the inclusion of timestamp data in log messages. On top of that, we observed the time zones implemented by the application in the process of recording logs.

It is usually recommended to standardize timestamp in UTC (Coordinated Universal Time) for logging across all systems. This approach aligns with ISO 8601 standards [43], which prescribe the format for timestamps and time zones. According to these standards, timestamps should either be set to UTC or to local time with an offset to UTC. Adopting this practice ensures consistency, reduces complexity in analysis, and assists in accurately correlating events.

To confirm the presence of timestamps and timezone information we executed the applications and observed the logs generated during their operation. During this process, we also checked if logs related to user actions were available. Overall, these steps offer a systematic approach to determine whether timestamp data is present in the logs of the applications under study.

Unique Identifier

The presence of unique identifiers allow analysts to filter log entries and focus on specific aspects of interest but also to connect a certain action with other logs or events collected elsewhere. For instance, a log entry that records a specific name of file being accessed by a specific user account can be linked to a network log entry which shows the corresponding user connection being established from the specific IP address.

Listing 1: Implementation of BeeBeep's log in source code

```
//BeeBEEP code snippet
```

```
bool Core::downloadFile( VNumber user_id, const FileInfo& fi, bool show_message )
{
   ...
   qDebug() << "Downloading file" << qPrintable( fi.path() ) << "from user" <<
       qPrintable( u.path() );
   mp_fileTransfer->downloadFile( u.id(), fi );
   return true;
}
```

The code excerpt from BeeBEEP in Listing 1 is used to log any successful file downloads within the application. The code uses the qDebug() function to print a log message at the debug level, which includes the file path and the user's path. This information is useful for detecting specific events, such as the downloading file and execution of potentially malicious files. By keeping track of this data, the application can detect and mitigate any security risks to such file path or user's path.

Listing 2: Handling successful and failed download events in Deluge

```
//Deluge code snippet
def on_download_fail(failure):
    log.warning(
        'Error occurred downloading file from "%s": %s',
        url,
        failure.getErrorMessage(),
    )
    result = failure
    return result

def on_download_success(result):
    log.debug('Download success!')
    return result

d = _download_file(
```

```
        url,
        filename,
        callback=callback,
        headers=headers,
        force_filename=force_filename,
        allow_compression=allow_compression,
        handle_redirects=handle_redirects,
    )
    d.addCallbacks(on_download_success, on_download_fail)
    return d
```

Listing 2 showcases an excerpt from Deluge, a Python-based torrent application. The `on_download_success` function acts as a callback, triggered by the deferred object resulting from the `_download_file` function upon successful file download. In this instance, when `on_download_success` is invoked, the application logs a debug message noting the download is success and then returns the input result argument. Merely logging a message like "Download success!" lacks detailed information about the download, rendering it challenging to correlate events with specific actions. It is crucial to incorporate more specifics, such as the file's name or download URL, to enhance the context of a successful download event.

On the other hand, the `on_download_fail` function performs a warning log with the URL and error message of the failure event, which provides more information about the failure and makes it easier to diagnose problems or understand the behavior of the system. With this information in hand, it is easier to determine the action and event identifier related to the logged event.

For this study, on the one hand we will investigate the number and percentage of log messages that contain three types of unique identifiers: file and path names, user identifiers, and network endpoints (host names and/or IP addresses). A high number of unique identifiers in application logs can ease the process of filtering log entries and focusing on specific aspects of event interest, which is crucial for security-related tasks.

To gather this information, we analyzed each extracted log message from the applications to identify the unique identifiers expected to be printed alongside the log message. This involves examining common name such as 'ipAddress', 'srcIP', 'url', etc., which typically represent unique network-related identifiers. For cases where naming conventions are not immediately identifiable,

we manually traced the unique identifier back to its source declaration in the application's source code. This approach allows us to accurately classify the type of unique identifier included in each log message.

We also count the amount of text-only log entries – i.e., those that report a fixed message without any parameter or identifier. Developers often use text-only messages because they are a simple and easy way to mark certain points of the execution (e.g., error in data processing) for debugging purposes. However, in the case of security-related tasks, text-only event logs may not provide enough granularity and context to detect security incidents or understand the relationships between log entries. A high percentage of text-only log entries can therefore signal an application whose logs are not suitable for event correlation and security investigations.

Partitioning Waypoints

Previous research has shown that by fusing both application and audit logs it is possible to achieve a high level of precision in identifying the origin of an attack [42]. This operation requires the presence of particular messages that act as waypoint to partition the provenance graph. Complex application consist of a set of units, such as separate windows, conversation threads, and processing of network connections [42, 39, 38]. A unit is made up of a series of transactions, each representing a single, sequential step in the execution of the unit. Therefore information such as the TabID, FolderID, messageID or chatID can be used to identify the current execution unit, which is otherwise difficult to extrapolate at the syscall level.

In order for application logs to be used for event partitioning, each execution unit performed and logged by the application should have a unique identifier to indicate the completion of an execution context and the start of another one.

Therefore, we will investigate the availability of any unique identifiers that can act as a waypoint. In particular, we investigate whether logs contain information about when the application starts to process a new document (or close an old one). For applications that do not involve files, we will examine the availability of logs containing unique identifiers related to the processing of new network connections or unique identifiers associated to new conversation threads. If these identifier are present, we consider that an analyst (or an automated system) could use them as waypoint to partition the activity of the application in separate units.

User Actions

Actions such as downloading an infected attachment, clicking a malicious link, (un)consciously changing settings, and allowing the application to execute untrusted documents can lead to security incidents. Thus, an applications should log any actions which can later be used to investigate or detect attacks or attempts to misuse the software. However, despite the importance of logging sensitive user actions (which is suggested by many guidelines and standards such as ISO 27002), there is no precise checklist on how developers should identify such actions.

In our experiments we propose a list of user actions that should be logged for each category of application, as summarized in Table 2. The list includes both the end-user's interactions with the application and the application's interactions with external environment. It is essential to note that this list is not exhaustive and only used as a starting point to understand the current state of application logs.

Log Levels

Log levels are used to configure the amount of recorded information and organize log data. For instance, `trace`, `debug`, and `info` levels usually provide detailed data that can be used for debugging and application performance analysis. On the other hand, `warn` and `error` level logs can be centralized to provide information on unusual events, which might be associated to security threats. From a security perspective, the presence of a log level can help to determine whether an event needs immediate investigation, could be deferred for a later analysis, or can be safely ignored. This prioritization, results in more efficient and effective security monitoring and analysis.

Therefore, in order to evaluate the effectiveness of application logs for detecting misuse, we will assess the default log-level configuration and the consistency and availability of log levels associated with specific user actions and events.

Exploitation

While system-level monitoring can be capable of detecting when an application is compromised, it is often insufficient to determine the cause (i.e., the input associated to the attack) which is a valuable information for post-incident analysis and incident response. In principle, application logs can help security analysts to better understand how the attack occurred and identify the specific actions that led to the security incident. This information can be used to identify the root cause and remediate vulnerabilities in the application. Furthermore, these logs can help in forensic

Table 2: List of user actions to log for security analysis across various application categories

Application Category	Actions to Log
File manager	Search queries Changes to file or document permissions File or document executions Archive operations
Chat and communication	Login activities Clicks on shared links Unique, randomized, non-identifiable IDs for conversation activities Attachment downloads
Office	File executions with embedded scripts URL clicks within documents Attempts to access restricted files Macro executions Changes to settings
Torrent and file transfer client	Files downloaded Seeding activities Search queries Peer connections
Screen capture	Screen capture activities Saved screen captures
Security/Antivirus	Login activities Acceptances to run quarantined files Setting changes (e.g., enabling/disabling scanners)
Text editor	Installation and usage of external plugins Link or URL clicks Document openings and edits
Utilities	Service starts Download or save activities Installation and usage of external plugins
Media Player	Opened files Data exports External subtitle imports Media play starts External plugin or add-on installations Runs of external plugins or add-ons
Window manager	Setting changes Session starts
Remote desktop	Login activities Remote connection starts Failed connections
Application launcher	Search queries Applications or documents launched Setting changes

investigations to understand how the attacker gained access to the system, the actions they took, and what data they managed to access.

Since logging all external inputs is impractical and embedding a custom anomaly-detection approach to identify unusual cases is extremely complicated and error-prone, it is difficult to asses whether application logs contain sufficient information to capture malicious inputs. As a first step, we will measure whether logs cover all exception blocks in the program and whether they also report the data that was responsible for the exception.

In addition, we will perform five case studies in which we will execute Proof of Concept (PoC) exploits collected from the Common Vulnerabilities and Exposures (CVE) database against vulnerable versions of five application in our dataset. After each successful exploitation, we will analyze the logs searching for any entry that can be used to detect and identify the attack. We will also evaluate the availability of these log entries and their relevance to incident response and forensic investigations.

Availability

The availability of log information depends on the application's logging configuration. Some applications provide debug logs or runtime logs by default to improve the application or to facilitate the process of reporting unexpected crashes to the developers. Others may only log errors and require users or administrators to manually edit configuration files to tune the amount of logged information. Finally, some applications do not enable logging by default nor provide options to start logs in the deployment.

To determine the default availability and configuration of logs for each application, our methodology begins with a thorough review of the application's documentation, where we specifically look for any references to the default availability of logs or or any user-configurable options for logging output. For applications where the documentation does not provide clear information on the log availability, or on the ability to configure logs, we proceed to run the applications in a host environment. This allows us to manually check for any log-related information or settings that might enabled or configurable.

5. Results

In this section, we present the results of our measurement. Table 3 provides an overview of the overall size and amount of logging statements in each application, along with the primary programming language and the library or framework (if any) used for logging.

As a rough indication of the amount of logging, the last column shows the ratio between the number of lines of code and the number of lines dedicated to logging. The value varies between 3.7% for the wire-desktop encrypted communication software to 0.03% for the musikcube application (with a median of 0.68% and a mean of 0.88%)

Our findings show that 11 of the studied applications provide application logs by default, while 49 did not, as indicated in Table B.6. However, among the applications that do not have the feature of logging enabled by default, many provide a configuration or options for users to enable logging. In fact, out of the 49 studied applications that did not provide logs by default at runtime, 47 offers users the capability to enable logging through configuration or options at runtime. This capability can be leveraged for debugging, system analysis, or even execution partitioning. However, we also found that the applications xpdf and XDM do not have the capability to produce runtime logs or provide any options to enable logging.

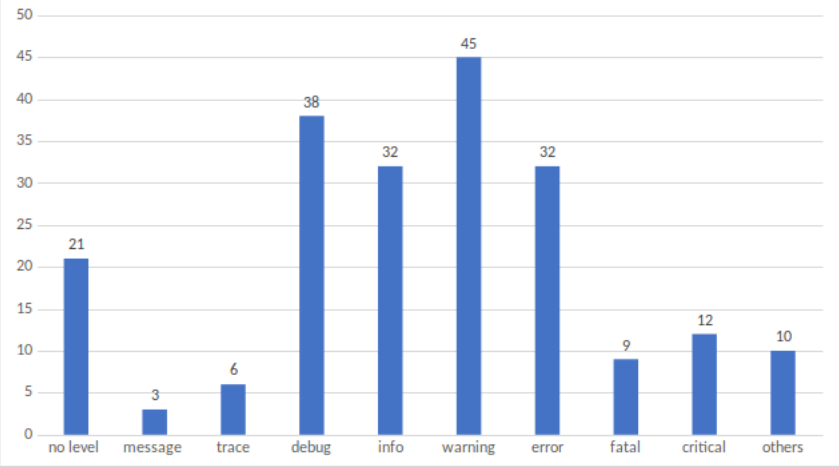

Figure 1: Usage of log level

Figure 1 shows the number of applications that use the different log levels. It is interesting to observe that 21 of them generated messages without any specific level. Warning is the most common

Table 3: Logs properties of studied applications.

Categories	**Applications**	**Logging utilities**	**Language**	**SLOC**	**LOLC**	**Percentage of log statement(%)**
File managers	Files - Nautilus	GLib Message Logging	C	98739	128	0.13
	Disks	GLib Message Logging	C	20724	68	0.33
	Dolphin	Qt - QMessageLogger	C++	40003	47	0.12
	nnn	Own logging facilities	C	11953	118	1.00
	recoll	Own logging facilities	C++	93972	1758	2.34
	Konqueror	Qt - QMessageLogger	C++	52191	370	0.72
	Thunar	GLib Message Logging	C	65835	105	0.16
Chat and Communication	IceChat(Windows)	Windows EventLog Class	C#	61981	261	0.42
	BeeBEEP	Qt - QMessageLogger	C++	61392	1147	1.89
	Pidgin	GLib Message Logging	C	169056	1320	0.78
	signal-desktop	Pino Logging	Typescript	208740	1384	0.66
	mattermost desktop	electron-log	Typescript	10945	224	2.05
	Konversation	Qt - QMessageLogger	C++	47393	259	0.55
	jitsi	SLF4J	Java	301383	2858	0.95
	wire-desktop	electron-log	Typescript	5677	212	3.73
	session-desktop	Bunyan logging API	Typescript	57841	609	1.05
	HexChat	Own logging facilities	C	62090	611	0.98
	Terminal- Irssi	GLib Message Logging	C	63755	440	0.69
	tox	Own logging facilities	C	48274	442	0.92
Office	Calligra	Qt - QMessageLogger	C++	660504	698	0.11
	Xournal++	GLib Message Logging	C++	52221	234	0.46
	SumatraPDF	Own logging facilities	C++	1059517	391	0.05
	zael	Qt - QMessageLogger	C++	13886	36	0.26
	Okular	Qt - QMessageLogger	C++	91398	434	0.52
	xpdf	Own logging facilities	C++	125529	1069	0.85
	ApacheOpenOffice	Own debug macro	C++	4591378	15235	0.33
	Scribus	Qt - QMessageLogger	C++	390442	515	0.13
	Evince-DocumentViewer	GLib Message Logging	C	81644	572	0.70
Torrent and file sharing platform	qBittorrent	Qt - QMessageLogger	C++	58813	437	0.74
	Deluge	Python logging facilities	Python	43956	1266	2.88
	transmissiongtk	Own logging facilities	C++	87299	193	0.22
	frostwire	Java logging utilities	Java	148930	942	0.63
	warpinator	Python logging facilities	Python	4111	134	3.26
Screen-capture	ShareX	.Net logging utilities	C#	115016	236	0.21
	OBS-Studio	Own logging facilities	C	340593	924	0.27
	Greenshot(Windows)	Log4net	C#	53293	684	1.28
Security or Antivirus	Seahorse	Qt - QMessageLogger	C	16901	88	0.52
	hashcat	Own logging facilities	C	198410	1519	0.77
	ClamAntiVirus(Linux)	Own logging facilities	C	215996	1492	0.69
Text editor	Zettlr	Own logging facilities	Typescript	23770	89	0.37
	gnome-text-editor	Qt - QMessageLogger	C	24903	48	0.19
	Vim	Own logging facilities	C	399870	140	0.04
	jrnl	Python logging facilities	Python	4000	22	0.55
Utilities	Console	GLib Message Logging	C	5856	23	0.39
	Cheese	Gstreamer debugging tool	C	6624	17	0.26
	Guvcview	Own logging facilities	C	30320	901	2.97
	XDM	Java basic log utilities	Java	27955	522	1.87
Media player	VLCMediaplayer	Own logging facilities	C	607435	454	0.07
	musikcube	Own logging facilities	C++	365945	122	0.03
	Kodi	spdlog	C++	732891	5619	0.77
Window manager	IceWM	Own logging facilities	C	70029	621	0.89
	tmux	Libevent	C	63656	647	1.02
	bspwm	Own logging facilities	C	12147	198	1.63
	Qtile	Python logging facilities	Python	38947	259	0.67
Remote desktop	Remotely	.Net Logging Basics	C#	58550	256	0.44
	xrdp	Own logging facilities	C	84133	1089	1.29
App launcher	albert	Qt - QMessageLogger	C++	4264	96	2.25
	launchy	Qt - QMessageLogger	C++	11735	33	0.28
	ulauncher	Python logging facilities	Python	6035	96	1.59
	Wox	Own logging facilities	C#	15367	150	0.98

level adopted by the applications in our dataset, but we also observed a number of unconventionally levels, such as 'silly', 'advice', and 'chatter', which we classified as 'others'.

Additionally, our analysis also revealed inconsistent log levels used for various user actions across the examined applications. For example, in the 'Utilities' category, the applications produced different log levels for the same user action. For instance, when a user starts the application, Cheese generates logs with the 'Info' level, while Guvcview and XDM produce logs without any level. This variability in log levels used by different applications can make it difficult to generalize the analysis, thus prolonging the process of security investigations.

5.1. Application Logs for Timeline Development

Only 31/60 of the applications in our study included timestamps in every log entry. Moreover, the implementation of timestamps varied among different application categories and is dependent on the implementation of the application's logging utilities. Of the 29 applications that do not include timestamps, 12 implemented their own logging utility, 9 used Qt's `QMessageLogging`, five `GLib` Message Logs, one the default `Java`, one with Windows Event Log and one with `.Net` logging libraries, as presented in Table B.6. It is interesting to observe that some applications do not include timestamps also when the logging frameworks they use support this feature. Even worse, in these case we were unable to find any configuration variable defined by the developers to include timestamps in all logs.

In our investigation of timestamp granularity, as presented in Table C.7, we observed that while most applications use a logging module, developers often define their own format for date and time.

Due to this flexibility and lack of standardized guidelines, applications can differ significantly in their timestamp presentations. The majority of the applications log timestamps up to the granularity of seconds. However, 17 of them extend this to milliseconds. Notably, tmux logs timestamps up to the microsecond level. We noted that hashcat logs the date and time solely at the commencement and conclusion of particular operations, like the initiation of a hashing process.

Regarding the availability of date data, 11 applications lacked date information. 20 logged both the month and date, but 12 out of 31 did not include the year in their logs. While the exact year might not always be crucial for security analysis, this inconsistency in timestamp presentation could complicate log analyses, especially when constructing detailed timelines using application logs.

According to our data, only four applications set their timestamp timezone to UTC (Coordinated

Universal Time), while the others use the system's local timezone for storing log entries. Storing logs in the UTC format or local time with an offset to UTC is a widely adopted practice, in line with ISO 8601 standards, in both software development and system administration. This universal standard is not affected by changes in time zones or daylight saving time adjustments, making it a crucial consistency factor when dealing with systems or users across multiple time zones. This ensures that log timestamps remain uniform and unchanged across different geographical locations.

Furthermore, in the process of analyzing logs, particularly during incident response, having a single, consistent time reference like UTC simplifies the process, as there is no need to convert times from different time zones to understand the sequence of events.

This emphasizes the importance of providing developers with clear guidelines for implementing timestamps in application logs.

5.2. Application Logs for Event Correlation

Figure F.2 shows that 77% of the applications in our dataset contain more text-only log messages (i.e., hardcoded strings with no variable parts) than messages containing user, file, and network identifiers combined. Text-only messages provide limited information about the logged events, diminishing the effectiveness of logs for practical investigation.

Table A.5 shows the actual amount of application log messages that contain unique identifiers related to users (username, account IDs, or nick names), network (IP addresses, domain names, and URIs), and file information (file and directory names). The value varied considerably among the applications in our dataset. For instance, 15.4% of the log messages of BeeBEEP report the associated user identifier, 17.4% of the messages of Konversation contains unique network identifiers, and 36.1/% of the messages of Zael have an associated filename.

Conversely, 11 applications didn't record any network-related unique identifiers (UIDs) in their logs and four applications didn't log any file UIDs at all. However, the decision to log UIDs associated with network or files largely depends on an application's core functionality and in some cases it might not be necessary or not apply at all. Applications categorized under File Manager, Office, and Security/Antivirus often integrate features like cloud functionality, updates via network, cloud-based storage, or collaboration tools. Specifically, for Security and Antivirus applications, logging is indispensable for tracing malicious activities, updating databases, or connecting to central databases.

If we look at the numbers of log statement in application, the worst in our dataset are the Console (a terminal emulator) and Cheese (a webcam tool to capture photos and videos) applications, whose logs are very limited (respectively only 23 and 17 log statements in their source code) and contain no unique identifiers at all across the three areas. This prevents any possible correlation with other sources of information and severely limit the use of these logs for forensic purposes. Finally, it is important to stress the fact that our study only assesses the broad availability of unique identifiers and further research is necessary to understand the usefulness of these UIDs in actual investigations.

5.3. Application Logs for Execution Partitioning

Application-generated logs can offer valuable insights on the usage patterns and behaviors of users, as well as detailed information about the inner workings of the application, including its interactions with other systems. Previous research has shown that these logs can be effectively leveraged for execution partitioning to derive precise attack provenance graphs. Logs that are not enabled at runtime may not be accessible for security analysis, and this can greatly decrease the possibility of security monitoring and analysis.

Previous research on utilizing application logs for execution partitioning highlighted the importance of including unique identifiers such as file names, IPs, or URIs to represent object of data in a single transaction of logs. These types of information can be leveraged to identify or gain insight into the execution recorded by the application. Thus, we examined the presence of unique identifiers in the process of opening or closing a new file or establishing new network connection or disconnecting from the network as a way to indicate the beginning of a new execution unit in the logs. The outcomes derived from utilizing application logs for execution partitioning can be found in Table B.6.

Our research revealed that out of the examined applications, 37 produced log entries containing at least one unique identifier associated to the ongoing execution unit. The distribution of these unique identifiers varied among the different application categories. For instance, among the file management applications, only one included a unique identifier in its log entries. Meanwhile, two out of the applications in the chat and communication category did not include any unique identifiers. In contrast, all of the applications in the screen capture, media player, and remote desktop categories contained at least one unique identifier in their log entries.

Listing 3: Pidgin log entries

```
--
(16:06:22) account: Connecting to account ###@irc.###.com.
(16:06:22) connection: Connecting. gc = 0x556421bc17d0
(16:06:22) dnsquery: Performing DNS lookup for irc.ubuntu.com
(16:06:22) Session Management: Received first save_yourself
(16:06:22) dns: Created new DNS child 5341, there are now 1 children.
---
---
(16:11:22) prefs: /pidgin/conversations/toolbar/wide changed, scheduling save.
(16:11:22) gtkconv: setting active conversation on toolbar 0x55df14133d90
(16:11:28) util: Writing file prefs.xml to directory /###/###/.purple
(16:11:28) util: Writing file /###/###/.purple/prefs.xml
(16:11:31) gtkconv: setting active conversation on toolbar 0x55df13ae2000
(16:11:32) gtkconv: setting active conversation on toolbar 0x55df14133d90
(16:11:33) server: Leaving room: [name of chat room]
(16:11:33) gtkconv: setting active conversation on toolbar 0x55df13ae2000
(16:11:33) irc: Got a PART on [name of chat room], which doesn't exist -- probably closed
(16:11:39) util: Writing file blist.xml to directory /###/###/.purple
(16:11:39) util: Writing file /###/###/.purple/blist.xml
(16:11:42) roomlist: unreffing list, ref count now 0
(16:11:42) roomlist: destroying list 0x55df13f35580
(16:11:44) account: Disconnecting account ###@irc.###.com (0x55df139226f0)
(16:11:44) connection: Disconnecting connection 0x55df13d97a60
(16:11:44) connection: Deactivating keepalive.
(16:11:44) connection: Destroying connection 0x55df13d97a60
(16:11:44) certificate: CertificateVerifier tls_cached unregistered
(16:11:44) certificate: CertificateVerifier singleuse unregistered
(16:11:44) certificate: CertificatePool tls_peers unregistered
(16:11:44) certificate: CertificatePool ca unregistered
(16:11:44) main: Unloading normal plugins
(16:11:44) plugins: Unloading plugin Message Notification
---
```

Listing 3 shows a Pidgin log entries when user make login to the application, changing the preference of the application, leaving chat room and logging out from the chat application. The logs generated by the applications provide a comprehensive view of the actions taken within the application, including user login information, file save events, and changes in preferences. For instance, there are a log entries shows changes in preference for the chat program (prefs: /pidgin/conversations/toolbar/wide changed, scheduling save), leaving a chat room with the name of room (server: Leaving room: [name of chat room]), and connecting and disconnecting from the account. The log also shows that the program is writing and saving files, such as `prefs.xml` and `blist.xml`, to the user's file directory.

As we look at the log entries, it shows that for each new unit of execution, there are log entries that contain unique identifiers such as the application account UID when user connections were made or server names when user connected to chat server. Additionally, the application logs provide clear execution units for every log entry, making it easier to identify and categorize actions. For example, actions related to a user's account will be categorized as 'account' and actions related to a server will be categorized as 'server'. The clear representation of this log entries can facilitate the process of parsing and partitioning of execution units.

On the other hand, Listing 4 shows instead the negative example of Calligra's logs, where entries are primarily related to errors and configuration issues. These entries indicate problems like missing icon themes, inability to access files or directories, and challenges initializing features or setting shortcuts. Though these log details are essential for enhancing the application's performance, they don't help in execution partitioning because they lack data on the made execution units. To use this log for partitioning, one would need a deeper analysis to spot usage patterns and object allocation in the application.

Listing 4: Caligra log entries

```
---
Icon theme "breeze" not found.
connect failed: No such file or directory
Hspell: can't open /usr/share/hspell/hebrew.wgz.sizes.
kf.sonnet.clients.hspell: HSpellDict::HSpellDict: Init failed
CalloutPathTool::CalloutPathTool(KoCanvasBase*) QAction(0x55e2cc8079b0 text="To Path"
    toolTip="To Path" shortcut=QKeySequence("P") menuRole=TextHeuristicRole visible=true)
```

```
kf.xmlgui: Shortcut for action "object_order_raise" "&Raise" set with
    QAction::setShortcut()! Use KActionCollection::setDefaultShortcut(s) instead.
kf.xmlgui: Shortcut for action "object_order_lower" "&Lower" set with
    QAction::setShortcut()! Use KActionCollection::setDefaultShortcut(s) instead.
kf.xmlgui: Shortcut for action "object_order_front" "Bring to &Front" set with
    QAction::setShortcut()! Use KActionCollection::setDefaultShortcut(s) instead.
---
```

5.4. Application Logs for Misuse Detection

To understand the use of application logs for misuse detection, we have compiled a list of user actions in different categories of applications. The list includes both events related to the end-user's interactions with the application and the events related to the application's interactions with the external environment. This list of actions guide us in identifying what the application should log to enable post-attack analysis. Logging user actions can provide a record of what actions were taken within application, allowing to identify patterns of behavior that may indicate misuse, such as repeated access to unauthorized resources. We performed each action on the studied applications and analyzed the logs they produced. The results as presented in Table D.8.

Our analysis of the log content and levels revealed inconsistencies among the applications in the same category when it came to logging user actions. For example, in the chat and communication category, BeeBEEP was found to generate detailed logs of all studied user actions at the Debug log level, while Irrsi-Client and Jitsi did not produce any logs for the tested user actions.

In particular, as part of our testing of chat and communication applications, we examined whether users were given the options to record or log conversations. All five applications we tested in this category provided the option for users to save conversation logs on their host computers. This can be useful for future reference or to recall important details from earlier conversations. However, this raises significant privacy concerns as these logs are stored in plain text and include names and body of messages. On the other hand, we discovered that most user actions which can be used for forensic purposes, such as downloading files and clicking on shared links, are also logged in the conversation log.

Moreover, in the office application category, most of the studied applications did not generate any logs regarding user actions. Okular was the only exception as they producing a 'Debug' log

when a user clicks on an embedded URL.

In the category of office applications, the majority of the applications we studied did not log any user actions. Okular was the only exception, producing a 'Debug' log when a user clicks on an embedded URL.

5.5. Application Logs for Attack Detection

As we discussed in the previous section, it is very difficult for an application to detect exploitation attempts. One aspect that, while insufficient, might help in this regard is to log any anomalous or error condition. Logging, particularly at these detected error junctures, is crucial as it sheds light on potential issues before they manifest as overt failures [44]. One way for programs to detected unexpected errors is through exceptions and therefore we believe that logging these conditions (and the data that might have been responsible to trigger them) can be a first step for an applications to record malicious undertakings or potential attacks. Thus, a consistent logging approach in exception blocks yields profound insights into an application's behavior, facilitating the identification of any unusual patterns suggestive of security threats or breaches.

However, Table E.9 reveals that only 25 out of 60 applications implemented exception handling, due to the fact that exceptions are not supported in all programming languages. Among them, Open Office has the highest number of exception blocks. GreenShot is instead the application with the largest fraction (77.4%) of logging statements in exception blocks, followed by Mattermost-Desktop at 74.51%.

Only five application (Transmission, Ulauncher, IceChat, Warpinator, and Wox) logged all exception statements, aiding in providing valuable information for troubleshooting and resolving issues. In contrast, three apps (Jrnl, Qtile, and Recoll) did not produce any log related to exceptions. We conclude that this inconsistency is due to a lack of standard guidelines for logging in application development.

Whether application logs are sufficient to detect and investigate a successful exploitation is a very complicated question, as it vastly depends on the type of vulnerability and the exploit itself. Therefore, it is difficult to answer this point by simply looking at the log messages themselves. Instead, we decided to conduct a separate study in which we selected five applications in our dataset and tested public exploits against them. The application version, platform, and exploitation type, are shown in Table 4.

Table 4: The result of the logs generated during the exploitation of selected applications

Applications	Version	Tested Platform	Type of exploitation
Evince	3.17.1	Ubuntu 16.04.6	Execution of arbitrary command
Nautilus	3.4.2	Kali 1.0.6	Execution of arbitrary command
LibreOffice Writer	6.2.5	Ubuntu 18.04	Directory traversal attack
VLC Media Player	2.2.8	Windows 10	Execution of arbitrary code via MKV files
HexChat	2.10.0	Ubuntu 22.04	Directory traversal attack

Our study indicates considerable variance in the utility of application logs for attack detection, subject to the specific application and the nature of the attack. While three applications logged events during the exploitation, these logs lacked definitive indicators of successful breaches. The absence of logs that show any exploitation makes it harder for analysts to use them for detecting security threats. For example, in our study with VLC Media Player, even when we turned on the highest logging level, no logs were produced during an attack. This lack of logs, including user actions, highlights challenges in using application logs from VLC for security purposes. In contrast, with HexChat, when users enable logging, the application records vital details like the connected server's IP, connection status, and chat history. This can offer insights into potential threats. Yet, this logging is only active when a user permits it. Many might not do so due to privacy concerns.

6. Result Summary

Based on our analysis conducted on 60 popular desktop applications, we have determined that their logs are not well suited for forensic purposes. In fact, our findings showed that:

1. 29/60 of the applications did not include timestamps in their log entries, making it challenging to determine the chronological order of events.

2. More than half of the applications produce log messages consisting mainly in constant text strings. Compared to messages that contain identifiers like user, network, and file data, this emphasis on text-only logs complicates event correlation.

3. 23/60 of the applications did not log any unique identifier at the beginning of a new execution unit, making it challenging to define new execution units and objects involved in those units.

4. Inconsistent logging of specific user actions across different applications makes it difficult for application and security analysts to understand patterns of behavior that may indicate misuse.

5. 35/60 of the studied applications did not use exception handling. Out of the 25 remaining applications, only five applications thoroughly logged exception statements. Thus, valuable information about unexpected errors and potentially vulnerable features or code could go unnoticed.

6. Successful exploitation often does not result in any relevant log data, and even when such data is produced, it is not available by default.

In conclusion, the results of our study highlight several significant challenges in leveraging application logs for security analysis. Even when logs are enabled, the deficiency of timestamps in 29 of the studied applications, the high proportion of text-only log events in over half of the applications, and the inconsistent logging of user actions and unique identifiers across different applications make it difficult for security analysts in performing thorough assessments. These findings highlight the need for improved logging practices for better logging to aid security analysis and incident detection.

7. Discussion

In this section, we discuss the implications of our findings and possible future directions.

Needs of Application's Logs for Forensic. Our analysis of 60 popular open-source applications suggests that current logging practices of desktop software are largely inadequate, limiting the effectiveness of their logs for forensic purposes. In particular, our results show that while these logs are rich in details, they often lack critical data, such as consistent timestamps and unique identifiers, which are essential for effective forensic analysis. This findings highlight the necessity to align application logs with best practices already adopted in other sources, such as operating systems and networks.

Investigators frequently compile comprehensive super-timelines during incidents or attack investigations. These timelines typically involve data collected from a variety of sources, each offering distinct insights and perspectives, and contributing to a more extensive and fine-grained analysis.

It is crucial to understand that our research does not suggest to rely solely on application logs. Instead, it emphasizes their potential use to complement existing data sources, thanks to the unique contextual information application logs can provide.

In fact, we believe the application logs can play a significant and specific role in the context of an investigation. The availability of their data can be crucial to fully understand the the interaction of users with external data or events – connecting the dots provided by other logs.

Sadly, due to their inadequacy, these logs are still under-represented in broader forensic studies. Understanding their limitations, which is the goal of our study, is crucial to mitigate this problem.

Solutions to Enhance Application Logging. Our research emphasizes a number of issues and inconsistencies with application logs and emphasizes the importance of proposing actionable solutions. Unfortunately, there are limited guidelines and practical solutions proposed in the literature that focus specifically on the forensic use case. Even worse, there is a lack of static analysis tools that can help developers to identify (and improve) logging issues during the development lifecycle. We believe that our taxonomy can help researchers to advance in both directions: to first create guidelines of *what* needs to be logged, and then to develop tools that can automatically enforce or verify that the implementation follows these guidelines.

Certain things are easy to fix, such as inconsistencies in the timestamp formats or the use of predefined log levels. These cases can be fixed by adopting a proper and standardized log framework or library. Other issues, like the lack of unique identifiers or the absence of log statements that precisely report error and exceptions, are more complex. Finally, some of the aspects are specific to a certain class of applications and thus need to be addresses on a case-by-case basis. This is the case, for instance, of the need to report when an application parses new external and user-provided data or when (and how) it interacts with the rest of the environment.

Finally, new forensic tools need to be developed to take advantage of this information and help investigators to combine and correlate application logs with other sources.

Overall, the collaboration between software engineering, cybersecurity, digital forensics, industry, and academia is essential for developing these guidelines and tools.

8. Conclusion

It is important for application logs to be ready for security analysis and forensic investigation, ensuring that critical information about application events is captured, preserved, and available for use. Therefore, this study highlights the challenges faced in utilizing application logs for security analysis. Application logs play a critical role in providing essential information about events between users and applications, but their use in security analysis has been understudied. This study significantly contributes to the understanding of the challenges in using application logs for security analysis. The findings indicate that the current logging practices of 60 open-source applications are inadequate for meeting common forensic tasks, such as timeline development, event correlation, execution partitioning, misuse detection, and attack detection. The findings reveal the need for including timestamps in logs, improving the inclusion of unique identifiers in log entries, and ensuring consistent logging of specific user actions. Additionally, the study highlights the importance of including data on successful exploitation in logs for detecting and preventing future attacks. The significance of this study lies in providing insights into the challenges of using application logs for security analysis and the necessity to improve application logging practices for security purposes. By enhancing the quality of application logs, security analysts and forensic investigators can more effectively monitor application events, detect security incidents, and respond to them.

Through our analysis, we aimed to understand the quality and consistency of log data across different applications and to identify any potential issues or gaps in application logs for security. By providing an in-depth examination of the data provided by application logs, we aim to offer valuable insights into their potential as a tool for detecting and preventing security-related issues.

References

[1] Z. Li, A. R. Chen, X. Hu, X. Xia, Are They All Good? Studying Practitioners Expectations on the Readability of Log Messages, 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023) (2023).

[2] S. Locke, H. Li, T.-H. Chen, W. Shang, W. Liu, LogAssist: Assisting Log Analysis Through Log Summarization, IEEE Transactions on Software Engineering 48 (9) (2022) 3227–3241. doi:10.1109/TSE.2021.3083715.
URL https://ieeexplore.ieee.org/document/9442364/

[3] Z. Li, C. Luo, T.-H. Chen, W. Shang, S. He, Q. Lin, D. Zhang, Did We Miss Something Important? Studying and Exploring Variable-Aware Log Abstraction, in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, Melbourne, Australia, 2023, pp. 830–842. doi:10.1109/ICSE48619.2023.00078.
URL https://ieeexplore.ieee.org/document/10172652/

[4] Y. Huo, Y. Li, Y. Su, P. He, Z. Xie, M. R. Lyu, AutoLog: A Log Sequence Synthesis Framework for Anomaly Detection, arXiv:2308.09324 [cs] (Aug. 2023).
URL http://arxiv.org/abs/2308.09324

[5] J. King, R. Pandita, L. Williams, Enabling forensics by proposing heuristics to identify mandatory log events, in: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, HotSoS '15, Association for Computing Machinery, New York, NY, USA, 2015. doi:10.1145/2746194.2746200.

[6] F. Rivera-Ortiz, L. Pasquale, Towards automated logging for forensic-ready software systems, in: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), 2019, pp. 157–163. doi:10.1109/REW.2019.00033.

[7] F. Rivera-Ortiz, L. Pasquale, Automated modelling of security incidents to represent logging requirements in software systems, in: Proceedings of the 15th International Conference on Availability, Reliability and Security, ARES '20, Association for Computing Machinery, New York, NY, USA, 2020. doi:10.1145/3407023.3407081.

[8] D. Wheeler, Understanding seventeenth-century ships logbooks: An exercise in historical climatology, Journal for Maritime Research 6 (1) (2004) 21–36. doi:10.1080/21533369.2004.9668335.
URL http://www.tandfonline.com/doi/abs/10.1080/21533369.2004.9668335

[9] Payment card industry (pci) data security standard requirements and security assessment procedures version 3.2.1 (2018).

[10] Five guidelines for robust logging — hackernoon.
URL https://hackernoon.com/five-guidelines-for-robust-logging

[11] Logging best practices - dzone performance.
URL https://dzone.com/articles/logging-best-practices

[12] Logging best practices - dev community.
URL https://dev.to/raysaltrelli/logging-best-practices-obo

[13] Logging guidelines for developers - diwebsity.
URL https://www.diwebsity.com/2016/04/05/logging-guideliness/

[14] Logging guidelines.
URL https://developer.atlassian.com/server/confluence/logging-guidelines/

[15] Logging best practices: The 13 you should know — dataset.
URL https://www.dataset.com/blog/the-10-commandments-of-logging/

[16] Developer's guide - logging - best practices.
URL https://doc.onloupe.com/Logging_BestPractices.html

[17] OWASP, Logging - owasp cheat sheet series.
URL https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

[18] A. Chuvakin, G. Peterson, How to do application logging right, IEEE Security and Privacy 8 (4) (2010) 82–85. doi:10.1109/MSP.2010.127.

[19] D. Brouwer, P. Mertens, Forensic logging requirements (2015).
URL https://www.compact.nl/articles/forensic-logging-requirements/

[20] E. Casey, S. Barnum, R. Griffith, J. Snyder, H. Van Beek, A. Nelson, Advancing coordinated cyber-investigations and tool interoperability using a community developed specification language, Digital Investigation 22 (2017) 14–45. doi:10.1016/j.diin.2017.08.002.
URL https://linkinghub.elsevier.com/retrieve/pii/S1742287617301007

[21] E. Casey, S. Barnum, R. Griffith, J. Snyder, H. Van Beek, A. Nelson, The Evolution of Expressing and Exchanging Cyber-Investigation Information in a Standardized Form, in: M. A. Biasiotti, J. P. Mifsud Bonnici, J. Cannataci, F. Turchi (Eds.), Handling and Exchanging Electronic Evidence Across Europe, Vol. 39, Springer International Publishing, Cham, 2018, pp. 43–58, series Title: Law, Governance and Technology Series. doi:10.1007/978-3-319-74872-6_4.
URL http://link.springer.com/10.1007/978-3-319-74872-6_4

[22] E. Casey, A. Nelson, J. Hyde, Standardization of file recovery classification and authentication, Digital Investigation 31 (2019) 100873. doi:10.1016/j.diin.2019.06.004.
URL https://linkinghub.elsevier.com/retrieve/pii/S1742287618304602

[23] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, T. Xie, Where do developers log? an empirical study on logging practices in industry, in: Companion Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 24–33.

[24] Z. Li, T. H. Chen, W. Shang, Where Shall We Log? Studying and Suggesting Logging Locations in Code Blocks, Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020 (2020) 361–372doi:10.1145/3324884.3416636.

[25] M. Cinque, D. Cotroneo, A. Pecchia, Event logs for the analysis of software failures: A rule-based approach, IEEE Transactions on Software Engineering 39 (6) (2013) 806–821. doi:10.1109/TSE.2012.67.

[26] K. Yao, G. B. de Pdua, W. Shang, C. Sporea, A. Toma, S. Sajedi, Log4perf: Suggesting and updating logging locations for web-based systems performance monitoring, Empirical Software Engineering 25 (1) (2019) 488531. doi:10.1007/s10664-019-09748-z.

[27] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, D. Zhang, Learning to log: Helping developers make informed logging decisions, Proceedings - International Conference on Software Engineering 1 (2015) 415–425. doi:10.1109/ICSE.2015.60.

[28] R. Ding, H. Zhou, J. G. Lou, H. Zhang, Q. Lin, Q. Fu, D. Zhang, T. Xie, Log2: A cost-aware logging mechanism for performance diagnosis, 2015.

[29] H. Li, W. Shang, A. E. Hassan, Which log level should developers choose for a new logging statement?, Empirical Software Engineering 22 (2017) 1684–1716. doi:10.1007/s10664-016-9456-2.

[30] P. He, S. He, Z. Chen, M. R. Lyu, Characterizing the natural language descriptions in software logging statements, ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (2018) 178–189doi:10.1145/3238147.3238193.

[31] Z. Liu, X. Xia, D. Lo, Z. Xing, A. E. Hassan, S. Li, Which variables should i log? (2021). doi:10.1109/TSE.2019.2941943.

[32] D. Yuan, S. Park, Y. Zhou, Characterizing logging practices in open-source software, Proceedings - International Conference on Software Engineering (2012) 102–112doi:10.1109/ICSE.2012.6227202.

[33] B. Chen, Z. M. Jiang, Characterizing and detecting anti-patterns in the logging code, 2017. doi:10.1109/ICSE.2017.15.

[34] B. Chen, Z. M. J. Jiang, Characterizing logging practices in java-based open source software projects a replication study in apache software foundation, Empirical Software Engineering 22 (2017). doi:10.1007/s10664-016-9429-5.

[35] S. Kabinna, C. P. Bezemer, W. Shang, M. D. Syer, A. E. Hassan, Examining the stability of logging statements, Vol. 23, 2018. doi:10.1007/s10664-017-9518-0.

[36] A. Pecchia, M. Cinque, G. Carrozza, D. Cotroneo, Industry Practices and Event Logging: Assessment of a Critical Software Development Process, Proceedings - International Conference on Software Engineering 2 (2015) 169–178. doi:10.1109/ICSE.2015.145.

[37] J. King, L. Williams, Log your crud: Design principles for software logging mechanisms, in: Proceedings of the 2014 Symposium and Bootcamp on the Science of Security, HotSoS '14, Association for Computing Machinery, New York, NY, USA, 2014. doi:10.1145/2600176.2600183.

[38] W. U. Hassan, M. A. Noureddine, P. Datta, A. Bates, Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis, Internet Society, 2020. doi:10.14722/ndss.2020.24270.

[39] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, D. Xu, {MPI}: Multiple perspective attack investigation with semantic aware execution partitioning, in: 26th USENIX Security Symposium (USENIX Security 17), 2017, pp. 1111–1128.

[40] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, A. Bates, NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage, in: Proceedings 2019 Network and Distributed System Security Symposium, Internet Society, San Diego, CA, 2019. doi:10.14722/ndss.2019.23349.
URL https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_03B-1-3_UlHassan_paper.pdf

[41] M. N. Hossain, S. Sheikhi, R. Sekar, Combating Dependence Explosion in Forensic Analysis Using Alternative Tag Propagation Semantics, in: 2020 IEEE Symposium on Security and Privacy (SP), IEEE, San Francisco, CA, USA, 2020, pp. 1139–1155. doi:10.1109/SP40000.2020.00064.
URL https://ieeexplore.ieee.org/document/9152772/

[42] L. Yu, S. Ma, Z. Zhang, G. Tao, X. Zhang, D. Xu, V. E. Urias, H. W. Lin, G. Ciocarlie, V. Yegneswaran, A. Gehani, Alchemist: Fusing application and audit logs for precise attack provenance without instrumentation (2021). doi:10.14722/ndss.2021.24445.

[43] Iso 8601 date and time format, https://www.iso.org/iso-8601-date-and-time-format.html, accessed: January 7, 2024.

[44] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, S. Savage, Be conservative: Enhancing failure diagnosis with proactive logging, Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012 (2012) 293–306.

Appendix A. Percentages of User, Network, File, Open Connection and Open File Data in Studied Applications.

Table A.5 illustrates the distribution of logs by identifier, including user, network, and file logs, with the corresponding number and percentage of each type of log. Additionally, it shows the number and percentage of log entries that include open network connections and open files.

Table A.5: Number and percentage of user, network, file unique identifier in studied applications

Categories	**Applications**	**No. of User UID (Percentage)**	**No. of Network UID (Percentage)**	**No. of File UID (Percentage)**
File managers	Files-Nautilus	0 (0)	11 (8.59)	4 (3.13)
	Disks	0 (0)	2 (2.94)	5 (7.35)
	Dolphin	0 (0)	2 (4.26)	3 (6.38)
	nnn	0 (0)	0 (0)	7 (5.93)
	recoll	0 (0)	24 (1.37)	112 (6.37)
	Konqueror	0 (0)	36 (9.73)	41 (11.08)
	Thunar	0 (0)	4 (3.81)	7 (6.67)
Chat and Communication	IceChat(Windows)	2 (0.77)	30 (11.49)	14 (5.36)
	BeeBEEP	177 (15.43)	171 (14.91)	232 (20.23)
	Pidgin	65 (4.92)	74 (5.61)	40 (3.03)
	signal-desktop	34 (2.46)	36 (2.6)	59 (4.26)
	mattermost desktop	0 (0)	28 (12.5)	3 (1.34)
	Konversation	20 (7.72)	45 (17.37)	11 (4.25)
	jitsi	77 (2.69)	143 (5)	12 (0.42)
	wire-desktop	10 (4.72)	6 (2.83)	9 (4.25)
	session-desktop	0 (0)	28 (4.6)	3 (0.49)
	HexChat	18 (2.95)	44 (7.2)	28 (4.58)
	Terminal- Irssi	25 (5.68)	23 (5.23)	12 (2.73)
	tox	17 (3.85)	14 (3.17)	2 (0.45)
Office	Calligra	0 (0)	23 (3.3)	37 (5.3)
	Xournal	0 (0)	0 (0)	11 (4.7)
	SumatraPDF	0 (0)	7 (1.79)	57 (14.58)
	zael	0 (0)	0 (0)	13 (36.11)
	Okular	0 (0)	11 (2.53)	58 (13.36)
	xpdf	0 (0)	0 (0)	94 (8.79)
	ApacheOpenOffice	8 (0.05)	227 (1.49)	80 (0.53)
	Scribus	0 (0)	0 (0)	25 (4.85)
	Evince-DocumentViewer	51 (8.92)	17 (2.97)	37 (6.47)
Torrent and file sharing platform	qBittorrent	3 (0.69)	44 (10.07)	62 (14.19)
	Deluge	0 (0)	5 (0.39)	52 (4.11)
	transmissiongtk	3 (1.55)	28 (14.51)	32 (16.58)
	frostwire	0 (0)	50 (5.31)	53 (5.63)
	warpinator	0 (0)	37 (27.61)	19 (14.18)
Screen-capture	ShareX	2 (0.85)	22 (9.32)	27 (11.44)
	OBS-Studio	0 (0)	1 (0.11)	55 (5.95)
	Greenshot(Windows)	0 (0)	24 (3.51)	34 (4.97)
Security or Antivirus	Seahorse	0 (0)	0 (0)	1 (1.14)
	hashcat	0 (0)	0 (0)	110 (7.24)
	ClamAntiVirus(Linux)	9 (0.6)	67 (4.49)	153 (10.25)
Text editor	Zettlr	0 (0)	0 (0)	30 (33.71)
	gnome-text-editor	0 (0)	5 (4.17)	0 (0)
	Vim	0 (0)	4 (2.86)	10 (7.14)
	jrnl	0 (0)	0 (0)	5 (22.73)
Utilities	Console	0 (0)	0 (0)	0 (0)
	Cheese	0 (0)	0 (0)	0 (0)
	Guvcview	0 (0)	51 (5.66)	56 (6.22)
	XDM	2 (0.38)	20 (3.83)	21 (4.02)
Media player	VLCMediaplayer	0 (0)	9 (1.98)	10 (2.2)
	Kodi	297 (5.29)	506 (9.01)	442 (7.87)
	musikcube	0 (0)	10 (8.2)	8 (6.56)
Window manager	IceWM	58 (9.37)	12 (1.94)	54 (8.72)
	tmux	1 (0.15)	12 (1.85)	24 (3.71)
	bspwm	0 (0)	3 (1.52)	0 (0)
	Qtile	0 (0)	4 (1.54)	9 (3.47)
Remote desktop	Remotely	4 (1.56)	8 (3.13)	4 (1.56)
	xrdp	43 (3.95)	88 (8.08)	98 (9)
App launcher	albert	0 (0)	2 (2.08)	7 (7.29)
	launchy	0 (0)	1 (3.03)	4 (12.12)
	ulauncher	0 (0)	5 (5.21)	12 (12.5)
	Wox	0 (0)	6 (4)	35 (23.33)

Appendix B. Logs for Partitioning, Availability, Logs Configurable and Timestamped Entries

Table B.6 provides a detailed overview of various applications and their log entries for partitioning, default availability, configurable logs, and timestamped entries.

Table B.6: Execution partitioning, default availability, configurable logs, and timestamped entries

Categories	Applications	Execution Partitioning	Log available by default	Log configuration	Date and Time Data
File managers	Files - Nautilus	✗	✗	✓	✓
	Disks	✗	✗	✓	✗
	Dolphin	✗	✗	✓	✗
	nnn	✗	✗	✓	✗
	recoll	✓	✗	✓	✓
	Konqueror	✗	✗	✓	✗
	Thunar	✗	✗	✓	✗
Chat and Communication	IceChat(Windows)	✓	✓	✓	✗
	BeeBEEP	✓	✗	✓	✗
	Pidgin	✓	✗	✓	✗
	signal-desktop	✓	✓	✓	✓
	mattermost desktop	✓	✗	✓	✓
	Konversation	✓	✗	✓	✗
	jitsi	✓	✗	✓	✓
	wire-desktop	✓	✓	✓	✓
	session-desktop	✓	✗	✓	✓
	HexChat	✗	✗	✓	✓
	Terminal- Irssi	✗	✗	✓	✓
	tox	✓	✗	✓	✗
Office	Calligra	✗	✗	✓	✗
	Xournal++	✗	✗	✓	✓
	SumatraPDF	✓	✗	✓	✗
	zael	✓	✗	✓	✗
	Okular	✗	✗	✓	✗
	xpdf	✗	✗	✗	✗
	ApacheOpenOffice	✓	✗	✓	✗
	Scribus	✗	✗	✓	✗
	Evince-DocumentViewer	✗	✗	✓	✗
Torrent and file sharing platform	qBittorrent	✓	✗	✓	✓
	Deluge	✗	✗	✓	✓
	transmissiongtk	✓	✗	✓	✓
	frostwire	✓	✗	✓	✓
	warpinator	✓	✗	✓	✓
Screen-capture	ShareX	✓	✓	✗	✗
	OBS-Studio	✓	✓	✓	✓
	Greenshot(Windows)	✓	✓	✓	✓
Security or Antivirus	Seahorse	✗	✗	✓	✓
	hashcat	✗	✗	✓	✓
	ClamAntiVirus(Linux)	✓	✗	✓	✓
Text editor	Zettlr	✓	✗	✓	✓
	gnome-text-editor	✗	✗	✓	✓
	Vim	✓	✗	✓	✗
	jrnl	✓	✗	✓	✓
Utilities	Console	✗	✗	✓	✗
	Cheese	✗	✗	✓	✓
	Guvcview	✓	✗	✓	✗
	XDM	✗	✗	✗	✗
Media player	VLC Media Player	✓	✗	✓	✗
	musikcube	✓	✓	✓	✓
	Kodi	✓	✓	✓	✓
Window manager	IceWM	✓	✗	✓	✗
	tmux	✓	✗	✓	✓
	bspwm	✓	✗	✓	✗
	Qtile	✗	✓	✓	✓
Remote desktop	Remotely	✓	✓	✓	✓
	xrdp	✓	✗	✓	✗
App launcher	albert	✓	✗	✓	✗
	launchy	✓	✗	✓	✗
	ulauncher	✗	✗	✓	✓
	Wox	✓	✓	✗	✓
TOTAL		37	11	56	31

Appendix C. Analysis of Timestamp Granularity in Application Logs

Table C.7 showcases the granularity of date and timestamps in logs across various applications, offering insights into how different software systems prioritize time-precise logging.

Table C.7: Date and time granularity of logs across applications

Categories	**Applications**	**Year**	**Month**	**Date**	**Hour**	**Min**	**Second**	**Milisecond**	**UTC timezone**
File managers	Files - Nautilus	✗	✗	✗	✓	✓	✓	✓	✗
	recoll	✓	✓	✓	✓	✓	✓	✗	✗
Chat and Communication	signal-desktop	✓	✓	✓	✓	✓	✓	✓	✓
	mattermost desktop	✓	✓	✓	✓	✓	✓	✓	✗
	jitsi	✓	✓	✓	✓	✓	✓	✓	✓
	wire-desktop	✓	✓	✓	✓	✓	✓	✓	✗
	session-desktop	✓	✓	✓	✓	✓	✓	✗	✓
	HexChat	✗	✓	✓	✓	✓	✓	✗	✗
	Terminal- Irssi	✗	✗	✗	✓	✓	✓	✓	✗
Office	Xournal++	✗	✗	✗	✓	✓	✓	✓	✗
Torrent and file sharing platform	qBittorrent	✓	✓	✓	✓	✓	✓	✓	✓
	Deluge	✓	✓	✓	✓	✓	✓	✓	✗
	transmissiongtk	✓	✓	✓	✓	✓	✓	✗	✗
	frostwire	✓	✓	✓	✓	✓	✓	✗	✗
	warpinator	✓	✓	✓	✓	✓	✓	✗	✗
Screen-capture	OBS-Studio	✗	✗	✗	✓	✓	✓	✓	✗
	Greenshot(Windows)	✓	✓	✓	✓	✓	✓	✓	✗
Security or Antivirus	Seahorse	✗	✗	✗	✓	✓	✓	✓	✗
	hashcat*	✓	✓	✓	✓	✓	✓	✗	✗
	ClamAntiVirus(Linux)	✓	✓	✓	✓	✓	✓	✗	✗
Text editor	Zettlr	✗	✗	✗	✓	✓	✓	✗	✗
	gnome-text-editor	✗	✗	✗	✓	✓	✓	✗	✗
	jrnl	✗	✗	✗	✓	✓	✓	✗	✗
Utilities	Cheese	✗	✗	✗	✓	✓	✓	✓	✗
Media player	musikcube	✗	✗	✗	✓	✓	✓	✗	✗
	Kodi	✓	✓	✓	✓	✓	✓	✓	✗
Window manager	tmux**	✗	✗	✗	✓	✓	✓	✓	✗
	Qtile	✓	✓	✓	✓	✓	✓	✗	✗
Remote desktop	Remotely	✓	✓	✓	✓	✓	✓	✓	✗
App launcher	ulauncher	✓	✓	✓	✓	✓	✓	✗	✗
	Wox	✓	✓	✓	✓	✓	✓	✓	✗
TOTAL		19	20	20	31	31	31	17	4

* Indicates the application that logs the date and time solely at the start and conclusion of an action's execution.

** Refers to the application showcasing granularity up to the microsecond level.

Appendix D. Presentation of logs on user actions and their level

Table D.8 presents the availability of logs on user actions and their level for different applications. The table includes several categories of actions on different category of applications. Each user actions is listed with a "Yes" or "No" indicating whether they have the feature of logging user actions, and the level of the log is available in bracket. Dash (–) means the application does not provide the feature related to that user action. This table serves as a reference for understanding the availability of log entries for various user actions across different categories of applications then used to referred application logs for misuse detection.

Table D.8: Logs availability on user actions and their level

Categories	Applications	Search Query	Permission changes	Assessing File	Zip file activity
File managers	Files - Nautilus	No	No	No	No
	GNOME disks	–	–	–	–
	Dolphin	No	No	No	No
	nnn	No	–	Yes (No level)	Yes (No level)
	recoll	Yes (Debug)	–	Yes (Debug)	Yes (Debug)
	Konqueror	No	No	No	No
	Thunar	No	No	No	No

Categories	Applications	Login activity	Shared link is clicked	Anonymized IDs of conversation participant	File download activity
Chat and Communication	IceChat (Windows)	No	No	Yes (In conversation log)	No
	HexChat-Client	Yes (No level)	Yes (In conversation log)	Yes (In conversation log)	Yes (In conversation log)
	Terminal- Irssi	No	No	No	No
	BeeBEEP	Yes (Debug)	Yes (Debug)	Yes (Debug)	Yes (Debug)
	Pidgin	Yes (Notice)	Yes (In conversation log)	Yes (In conversation log)	No
	Signal-Desktop	No	No	No	No
	tox	Partial (Debug)	No	No	Partial (Debug)
	mattermost-desktop	No	Yes (Debug)	Yes (In conversation log)	No
	konversation	No	Yes (In conversation log)	Yes (In conversation log)	No
	jitsi	No	No	No	No
	wire-desktop	Partial (Info)	No	No	No
	session-desktop	Partial (Info)	No	Partial (Info)	Partial (Warn)

Categories	Applications	Executing files with embedded scripts	User clicking on a URL within a document	Attemped access a restricted file	Executing macros	Making changes to settings
Office	Scribus	No	–	No	–	No
	Calligra Words	No	No	No	–	No
	Xournal++	No	–	No	–	No
	LibreOffice Writer	No	No	No	No	No
	SumatraPDF	No	No	No	–	No
	zael	No	No	–	–	No
	Evince	No	No	No	–	No
	Okular	No	Yes (Debug)	No	–	No
	xpdf	No	No	No	–	–

Categories	Applications	Downloading file shared	File start seeding	Search query	Peer connection successful
Torrent and file sharing platform	qBittorrent	Yes (No Level)	No	–	–
	Deluge	No	No	–	–
	transmissiongtk	Yes (Info)	Yes (Debug)	–	–
	frostwire	Yes (Info)	No	Partial (Severe)	–
	warpinator	No	–	No	Yes (Info)

Categories	Applications	Start capturing screen	Saving captured screen
Screen-capture	ShareX	Partial (No level)	Yes (No level) - saved directory
	OBS-studio	Yes (Info)	Yes (Info)
	Greenshot	No	Partial (Info)

Table D.8 Continued: Logs availability and level

Categories	Applications	Login activity	Accept to run quarantine file	Changes in setting
Security or Antivirus	Seahorse/Keyring	No	–	No
	hashcat	–	–	–
	ClamAntiVirus(Linux)	–	No	No

Categories	Applications	Installing and using external plugin	Clicking links through the editor	Opening and editing documents
Text editor	Zettlr	No	–	The application does not allowed to run under root
	gnome-text-editor	No	No	No
	Vim	Partial (No level) - dir of plugin	No	Partial (no level)
	jrnl	–	No	The application does not allowed to run under root

Categories	Applications	Start application service	Document downloaded or saved	External plugin installed
Utilities	Console	No	–	No
	Cheese	Yes (Info)	No	–
	Guvcview	Yes (No Level)	Yes (No Level)	–
	XDM	Yes (No Level)	Yes (No Level)	–

Categories	Applications	Opened file	Exporting media	Importing subtitle	Playing media	Importing external plugins	Importing external plugins
Media player	VLC Media Player	Yes (Debug)	Yes (Debug)	Yes (Debug)	Yes (Debug)	Yes (Debug)	Yes (Debug)
	musikcube	Yes (Info)	No	–	Yes (Info)	No	No
	Kodi	Yes (Debug)	–	Yes (Debug)	Yes (Info)	Yes (Debug)	Yes (Debug)

Categories	Applications	Any setting changes	Start new session
Window manager	IceWM	Yes (Fail)	Yes (Message)
	tmux	No	Yes (Debug)
	bspwm	Yes (Error)	No
	Qtile	Yes (Error)	No

Categories	Applications	Login activity	Start remote connection	Failed connection
Remote desktop	Remotely	Yes (Info)	No	Yes (Warning)
	xrdp	Yes (Info)	Yes (Debug)	Yes (Error)

Categories	Applications	User search query	Name of application/-document launched	Changes in setting
App launcher	albert	Yes (Debug)	No	Yes (Debug)
	launchy	Yes (Debug)	Yes (debug)	No
	ulauncher	Partial (Debug)	Yes (Info)	Yes (Info)
	Wox	No	No	No

Appendix E. Exception Block Distribution, Logs within Blocks, and Logs with Exceptions

The table (referenced as Table E.9) displays the distribution of the number of exception blocks in the studied applications, the number of logs within those blocks, and the number of logs that contain exception data within the defined logs. The percentage representation of each data is included in parentheses.

Table E.9: Number of exception blocks, logs within those blocks and logs with exceptions

Categories	**Applications**	**Number of Exception Block**	**Number of Logs Statement in Exception (Percentage)**	**Number of Logs with Exception Info (Percentage)**
File managers	recoll	48	9 (18.75)	0 (0.00)
Chat and Communication	IceChat(Windows)	231	52 (22.51)	52 (100.00)
	signal-desktop	395	218 (55.19)	181 (83.03)
	mattermost desktop	51	38 (74.51)	32 (84.21)
	jitsi	2015	1219 (60.50)	1194 (97.95)
	wire-desktop	63	23 (36.51)	21 (91.30)
	session-desktop	225	104 (46.22)	83 (79.81)
Office	Calligra	71	23 (32.39)	10 (43.48)
	Xournal	43	23 (53.49)	17 (73.91)
	ApacheOpenOffice	4410	1379 (31.27)	1027 (74.47)
Torrent and file sharing platform	qBittorrent	50	14 (28.00)	13 (92.86)
	Deluge	450	21 (4.67)	4 (19.05)
	transmissiongtk	18	2 (11.11)	2 (100.00)
	frostwire	1352	344 (25.44)	341 (99.13)
	warpinator	83	26 (31.33)	26 (100.00)
Screen-capture	ShareX	193	112 (58.03)	109 (97.32)
	Greenshot(Windows)	323	250 (77.40)	247 (98.80)
Text editor	Zettlr	97	17 (17.53)	16 (94.12)
	jrnl	38	0 (0.00)	0 (0.00)
Utilities	XDM	342	187 (54.68)	183 (97.86)
Media player	musikcube	100	32 (32.00)	2 (6.25)
Window manager	Qtile	228	30 (13.16)	0 (0.00)
Remote desktop	Remotely	198	128 (64.65)	123 (96.09)
App launcher	ulauncher	52	6 (11.54)	6 (100.00)
	Wox	105	56 (53.33)	56 (100.00)

Appendix F. Log percentages with user, network, and file data, as well as text-only events.

The Figure F.2 provides a visual representation of the distribution of log percentages with user, network, and file data, as well as text-only events.

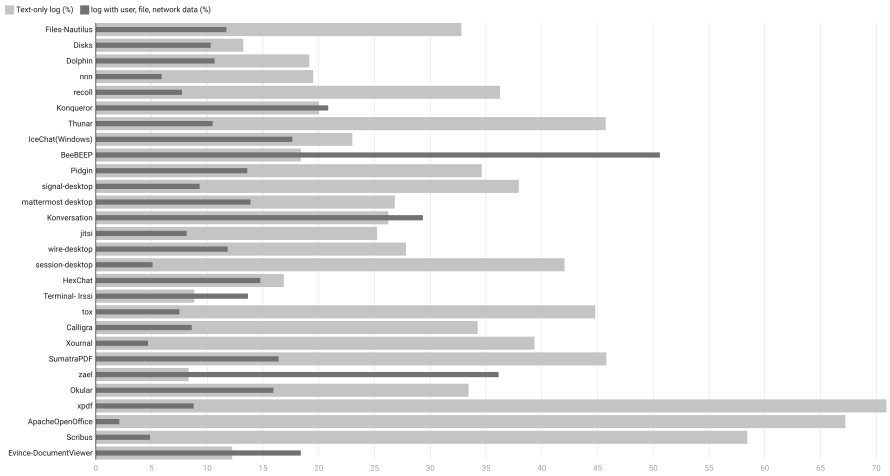

Figure F.2: The distribution of log percentages with user, network, and file data, as well as text-only events.

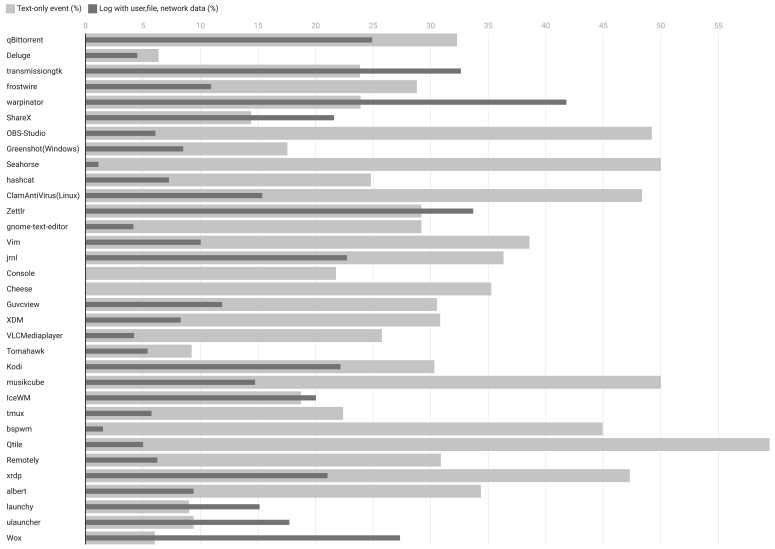

Figure F.2 Continued: The distribution of log percentages with user, network, and file data, as well as text-only events.