Forensic Science International: Digital Investigation 53 (2025) 301921

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS USA 2025 - Selected Papers from the 25th Annual Digital Forensics Research Conference USA

Uncovering linux desktop espionage

Lukas Schmidt [a,*], Sebastian Strasda [a], Sebastian Schinzel [a,b]

[a] Fraunhofer SIT, National Research Center for Applied Cybersecurity ATHENE, Rheinstraße 75, Darmstadt, 64295, Germany
[b] IT Security Lab, Münster University of Applied Sciences, Stegerwaldstraße 39, Steinfurt, 48565, Germany

ARTICLE INFO

Keywords:
Memory forensics
Espionage attacks
Malware
Linux/Unix

ABSTRACT

The increasing adoption of Linux-based desktop systems in various sectors, including critical infrastructures and personal use, has made them an attractive target for Advanced Persistent Threat (APT) groups and state actors. Yet, the espionage capabilities of Linux desktop malware and the forensic strategies for uncovering them remain largely unexamined. This paper addresses this gap by analyzing ten malware families that target the Linux desktop environment, studying the utilized espionage techniques, and introducing novel approaches to detect them using memory forensics.

Facing the multitude of espionage attack implementations that result from the diverse Linux desktop ecosystem, we propose to reduce the complexity of memory forensic investigations by focusing on the analysis of targeted core services. We evaluate our approach by implementing proof-of-concept Volatility plugins for identification of keylogging, screen capturing as well as camera and microphone recording malware, and prove their effectiveness by performing forensic analyses of real-world espionage techniques that were utilized during APT campaigns. Our evaluation shows that memory forensics is effective in uncovering Linux espionage attacks, and we are confident that our study provides valuable insights for future research and practical analysis of these threats.

1. Introduction

The continuous improvement of open source desktop software has led to a rise in the usage of Linux-based desktop systems. As a result, Linux-based workstations have been reported to be widely used by public institutions and found utilization even in critical infrastructures, e.g., at the U.S. Department of Defense (linux.com, 2007), India's Defence Ministry (Peri, 2023) or European law enforcement agencies (HILLENIUS, 2013).

Moreover, Linux-based desktop systems also gained popularity as personal computers for home use (Vaughan-Nichols). The economic potential of its large user base has been recognized by large game-developing companies like Steam (2024), which built a whole gaming ecosystem around the Linux desktop, as well as specialized companies offering professional-grade graphical applications (Case and Richard, 2017). Reflecting its wide adoption, Linux even overtook macOS' user share as a gaming platform since 2023 (Steam, 2024), underlining today's relevance of Linux-based desktops in the field.

Contrarily, and despite multiple APT groups (Talos, 2023; Cyble, 2024; Kaspersky, 2020; Ramamoorthy et al., 2024) and state actors (Amnesty International, 2020) repeatedly target the Linux desktop, malware abusing Linux desktop functionality did not receive much attention in research. We are unaware of any studies evaluating the malicious capabilities of Linux desktop malware or addressing the defense against them. As a result, the detection and forensic analysis of desktop espionage attacks remain unsolved problems.

In this paper, we present a two-step approach to address these challenges. Firstly, we identify malware samples that target the Linux desktop environment and systematically analyze the techniques used to commit espionage on desktop users. We show that the investigated malware samples utilize a multitude of techniques to spy on users, which particularly target the desktop ecosystem.

Secondly, facing the advanced evasion and obfuscation methods of malware used by APT groups, we propose to forensically examine malware infections using memory forensics (Case and Richard, 2017). While the broad range of utilized espionage techniques complicate the application of memory forensic analyses, we show that most of the spy attacks rely on functionality provided by a few core services. We use this insight to reduce the complexity of memory forensic approaches and develop proof-of-concept implementations that are capable of detecting

* Corresponding author.
E-mail address: lukas-schmidt@fh-muenster.de (L. Schmidt).

https://doi.org/10.1016/j.fsidi.2025.301921

espionage attacks. To foster future research and analyses, we make these implementations publicly available in the form of plugins for the Volatility framework (Volatility Foundation, 2025).

We evaluate our approach by re-implementing real-world desktop malware attacks, executing them on systems powered by eight popular Linux distributions, and performing forensic analyses using the developed plugins. Our evaluation shows that espionage attacks also compromise seemingly resilient desktops based on Wayland, and we demonstrate that memory-forensic analyses are effective in detecting them.

Summarized, our contributions are as follows:

- We present, to our knowledge, the first extensive study on Linux desktop malware. We compiled a dataset of ten malware families targeting the Linux desktop, reverse engineered their behavior, and report on the techniques used for conducting espionage attacks on desktop users.
- We present novel techniques for detecting these espionage attacks by analysing the process memory of display servers, audio providers and the Linux kernel.
- By targeting eight popular Linux distributions, we analyze the impact of espionage techniques on modern Linux desktops and evaluate our detection methods.
- To foster future research and evaluation, we publish an open-source repository (Muenster, 2025) with implementations of identified real-world attacks, Volatility plugins for detecting them, and a dataset containing memory dumps of infected systems.

2. Background & motivation

2.1. Linux desktop environment

The Linux desktop environment is a complex ecosystem that comprises various userland components working in tandem. At its core, the Linux desktop environment relies on a display server that is responsible for rendering graphics and handling of input events from devices such as keyboards and mice. The two most commonly used display servers protocols include the X Window System (X11) and Wayland. While X-based environments utilize the XOrg server (X11) as display server, Wayland-based environments use one of the many available Wayland compositors. Additionally, the Linux desktop environment depends on several other key components, in particular audio servers and the Video 4 Linux (V4L) subsystem. The audio server, nowadays typically implemented using PipeWire, manages audio streams, offers sound mixing capabilities, and interacts with audio hardware devices such as sound cards and speakers. V4L is a set of APIs and drivers that provide access to video capture devices such as webcams, TV tuners, and other multimedia peripherals, providing functions such as video recording, streaming, and playback.

2.2. Memory forensics

In recent years, the increasing sophistication of modern malware has led to a shift in digital forensic analyses (Case and Richard, 2017). Traditional methods of malware detection, which rely on signature-based scanning and log analysis, fall short against advanced threats that are heavily obfuscated, encrypted and primarily reside in volatile memory to avoid detection. As a result, analyses of volatile memory became a critical component of forensic investigations (Ligh et al., 2014), either during live investigations or when analyzing memory snapshots of potentially compromised systems. Thereby modern frameworks facilitate the inspection of kernel and process memory (Volatility Foundation, 2025), offering investigators a complete view into a system's state, provided that the semantic byte-level data can be correctly interpreted. Therefore, memory forensics provides insight into information that is otherwise inaccessible and can be used to scale detection and alerting of advanced malicious software (Manna et al., 2021).

2.3. Motivating example

The malware families analyzed in this paper were mainly used by state actors or APT groups, e.g., to spy on political dissidents after successfully infiltrating their end-user devices (Amnesty International, 2020). We assume that attackers have fully compromised the victim's system and are spying on the victim's activities and communication. To reach their goals, attackers make use of sophisticated malware capable of covering its presence, e.g., by hiding running malware processes (Ligh et al., 2014). Against this background, memory forensics analyses are particularly valuable, as they can be used to uncover hidden intrusions. While previously researched malware mainly abuses kernel functionality to perform espionage (Ligh et al., 2014; Case et al., 2022), the architecture of Linux desktop systems allows to implement spy attacks by targeting desktop components, e.g., display or audio servers (see Snippet 1).

```
d = XOpenDisplay(NULL);
if (d == NULL) {
    fprintf(stderr, "Unable to open X display\n");
    return EXIT_FAILURE;
}
XGetInputFocus(d, &focus, &revert);
XSelectInput(d, focus, KeyPressMask |
↪  KeyReleaseMask | FocusChangeMask);
```

Snippet 1: Code of a keylogger that registers for events of the X11 server.

As a result, these attacks remain undetected by existing memory forensic approaches, as they primarily focus on the evaluation of kernel structures (Case et al., 2022; Ligh et al., 2014; Maxwell et al., 2013). To close this gap, we propose to perform memory forensic analyses of targeted desktop services. In this way, we find indicators of malware infections, e.g., by identifying display server clients that demonstrate suspicious behaviour (see Snippet 1).

3. Uncovering linux desktop espionage

3.1. Methodology

To effectively detect espionage attacks, a comprehensive overview of the features and techniques implemented by malware is essential. In the following, we describe our methodology for compiling a dataset of Linux desktop malware, present the results of a thorough analysis of their implemented espionage features and introduce approaches for detecting them. To the best of our knowledge, we thereby present the first systematic overview of Linux desktop malware.

3.1.1. Collecting malware families & samples

In the absence of a publicly available dataset on Linux desktop malware, we have compiled a dataset based on generally available information. Specifically, we first utilized search engines, i.e., Google and DuckDuckGo, to identify relevant malware families. We thereby conducted targeted searches using keywords and phrases such as *linux desktop malware*, *linux spyware*, and *linux trojan*, and analyzed the results, consisting of various articles, blog posts, and threat intelligence reports. If an article indicated that a malware family targets the Linux desktop, we searched malware repositories on Github (GitHub, 2025) and databases such as MalwareBazaar (MalwareBazaar (2025) and Vx Underground (2025), and downloaded the corresponding samples for subsequent analyses. As seen in Table 1, this process allowed us to

Table 1
Identified Linux desktop malware families, implementation language, sample availability, associated APT groups, and year the malware has been reported to be used in the wild.

Malware	Language	Source Code	Binary Sample	APT group	Reported
HackingTeam	C/C++	✓		State Actor	2013
Mokes	C/C++		✓	–	2016
Fysbis	C/C++		✓	APT28	2016
EvilGnome	C/C++		✓	–	2019
FinSpy	C/C++		✓	State Actor	2021
Alchimist/ Insekt	Go		✓	–	2022
DeimosC2	Go	✓		Lazarus	2022
Empire	Python	✓		Turla	2022
Pupy	Python	✓		UTG-Q-010	2024
Sliver	Go	✓		APT29	2024

identify and collect samples of ten distinct malware families targeting the Linux desktop, including those not previously documented or analyzed in the academic literature.

3.1.2. Reverse engineering of espionage techniques

To evaluate the espionage capabilities of the identified malware families, we reverse engineered the collected samples with a focus on implemented spy techniques. The collected samples existed in two primary forms: binary executables and source code. Given the diversity of samples and languages involved, we found manual analyses preferable to automated attempts.

We analyzed the collected code and binary samples by employing a combination of manual code review and static and dynamic analyses with the help of *Ghidra* and *gdb*. As the binary samples utilized custom packing and obfuscation mechanisms to complicate malware analysis, we evaluated their packing schemes and unpacked them before further analysis. We then investigated the implemented espionage capabilities and identified various spy attacks targeting services of the Linux desktop environment. We elaborate on the identified capabilities and their implementations in the following sections and summarize the results in Table 2.

3.1.3. Detecting malware using memory forensics

As illustrated in Sec. 2, we propose to uncover espionage attacks by analyzing the memory of targeted desktop services, particularly by identifying state changes that indicate malicious activity. In this way, we can detect the malware's behavior by investigating the address space of a few desktop services, avoiding a costly analysis of all running processes. Furthermore, this approach works independently of the implementation of the espionage techniques, i.e., our approach is independent of the programming language used, as well as runtimes and shared libraries.

We take benefit of the fact that open source software can be created with debugging symbols. We use these symbols to analyze structures in process memory that are affected by the execution of espionage attacks, and identify changes to data fields that can be used as indicators of espionage attacks. Using these insights, we implement proof-of-concept Volatility (Volatility Foundation, 2025) plugins that can be used to uncover espionage attacks on Linux desktop systems. In the following, we elaborate on the identified espionage techniques and the approaches used to detect them in more detail.

3.2. Key- & mouselogging

3.2.1. Malware implementation

Expectedly, we found most of the desktop malware samples

Table 2
Linux desktop malware families, the APIs & techniques used to implement spy attacks, and the desktop services targeted.

Lang.	Library	API/Technique	Targeted Service	HackingTeam	Mokes	Fysbis	EvilGnome	FinSpy	Alchimist/Insekt	DeimosC2	Empire	PuPy	Sliver
Keylogger													
C	xlib	XSelectInput	Display Server	●								●	
C	xlib	XISelectEvents									●	●	
C	xlib	XQueryKeymap				●		●			●	●	
Mouselogger													
C	xlib	XQueryPointer	Display Server	●									
Screen Capture													
C	xlib	XGetImage	Display Server	●									
Py	python-mss	XGetImage										●	
Go	screenshot	xgb.GetImage								●			●
Go	xgb	xgb.GetImage							●				
C	Cairo	cairo_xlib_surface_create					●						
C	gdk	gdk_pixbuf_get_from_window						●					
C++	Qt	QScreen.grabWindow			●								
Camera Rec.													
C	stdlib	open	V4L2 Device	●				●					
Micro. Rec.													
C	PulseAudio	pa_simple_new	Audio Server	●	●		●						
C++	Qt	QAudioInput											
C	PortAudio	pa_initialize										●	

implement key- and mouselogging features, e.g., functionality to record keystrokes to steal passwords, sensitive information, and other confidential material. Thereby all investigated malware samples abuse functionality of the Xorg server for implementing these espionage attacks, as the X server also handles the processing of input events for the application. Malware applies multiple techniques for capturing keystrokes and mouse pointer events.

Firstly, some malware samples register at the X server to receive core input events, which represents the traditional X-server implementation of input handling. Thereby, a connection to the X server is established using the XOpenDisplay function, and the malware registers for notification of keypress events using the XSelectInput API (see Snippet 1). As a result, the X server queues keyboard input events and delivers them when the malware signals readiness by calling XNextEvent.

Secondly, some malware families use the more modern input extensions as an alternative for capturing input events. To do so, they create an event mask with XISetMask and utilize it to register for events with the XISelectEvents function. Similar to the handling of the core event queues, keypress events are delivered to the malware when requested with XNextEvent.

Thirdly, and differing from the previously introduced variants, some malware samples implement keylogging by fetching the logical state of the keyboard in microsecond intervals. After establishing a connection to the X server, the malware requests a bit vector containing the keyboard state with XQueryKeymap. The returned bit vector indicates key presses, which the malware compares to the bit vectors of previous intervals. Thereby, any changes between these vectors represent a keystroke.

We found that some malware samples utilize the same mechanisms to also query the mouse pointer's position, either by registering for the corresponding events or using the XQueryPointer function. Combined with the capture of screenshots, user's actions on the screen can be tracked accurately.

3.2.2. Memory forensic analysis

We shortly elaborate on the X server's internal memory structures to facilitate the understanding of the following approaches. The X11 server holds client information in a struct _Client, and stores pointers to each client's structure in a linked list called clients. Each _Client structure contains a pointer to the ClientIdPtr substructure, in which detailed information about the originating process is stored, such as the process ID and the process name. The X11 server uses screen and window abstractions to display a graphical user interface. A _Screen structure encapsulates information and resources related to a specific screen of a display, which can consist of one or more monitors. It contains a pointer to a linked list of top-level windows encapsulated in _Window structures, which holds all the relevant information required to manage and display a specific window.

3.2.2.1. Core events. When a malware requests core event notification using XSelectInput, the X11 server iterates over each window. It appends the malware client to a linked list named otherClients. This list holds _OtherClients structure instances, and each list member contains a mask. This mask is used to register the corresponding client for event notifications of the window and is evaluated when an input event, such as a key press, occurs. We leverage this fact to detect key- and mouseloggers with the *xevents* Volatility plugin. The plugin scans the heap of the X11 server for _Screen structure instances and uses screen instances found to identify the attached parent-level windows. Following, the plugin evaluates each window's otherClient list and identifies all clients that register for input event notification by setting the key press 0x03 or mouse event flags 0x02. Through this plugin, investigators can immediately identify key- and mouseloggers that rely on core event notification (see Fig. 1).

PID	Process	Client ID	Event Mask	Captured Events
1702	key_core	19	0x20003	KeyPress Events

Fig. 1. Xevents plugin detecting a keylogger that captures X11 core events.

3.2.2.2. Input extensions. By calling XISelectEvents(), malware registers for notification of events via input extensions. These behave similarly to the core event notifications but differ in implementation. In case a client wants to capture events via input extensions, the X11 server appends the client to a linked list holding instances of struct _InputClients, which is only accessible by following an extensive pointer chain: A pointer to this list can be found in the InputClients field of the OtherInputMasks structure, which can be accessed via a structure called _WindowOpt, which contains additional information about a window and itself is a substructure of struct _Window. We use this insight to implement key- and mouselogger detection in the *xinputextensions* plugin. The plugin identifies _Screen instances on the heap of the X11 server and iterates over the linked windows. The plugin follows the previously described pointer chain for each window and accesses the InputClients list. For each client in this list, the plugin evaluates a struct _XI2Mask, representing an array of masks. This array separately encodes event notification masks for each registered input device. To capture all input events, malware registers for the placeholder devices XIAllDevices or XIAllRawDevices and sets the flag 0x20 for keystroke logging or the flag 0x02 for mouse logging. By checking the masks for these values, the plugin can detect key and mouse capturing and effectively supports investigators in identifying malware that abuses input extensions (see Fig. 2).

The analysis of espionage attempts reading the logical state of input devices is identical to those used for screen capturing, and we describe their analysis below.

3.3. Screen capturing

3.3.1. Malware implementation

An espionage technique commonly used by malware is screen capturing, which involves saving an image of the screen to spy on sensitive data currently being displayed. While almost every investigated malware family provides screen capturing functionality, their implementations vary widely. As seen in Table 2, various shared libraries, packages, and APIs are utilized to acquire screenshots.

```
root = gdk_get_default_root_window();
root_width = gdk_window_get_width(root);
root_height = gdk_window_get_height(root);
/* ... */
screenshot = gdk_pixbuf_get_from_window(root, x,
↪  y, w, h);
```

Snippet 2: Screen capture utilizing the GDK APIs.

When investigating the source code of these modules, we found them to represent high-level wrappers around a few core libraries that talk to the system's display server (see Snippet 2). As the ability to capture screenshots is restricted in Wayland-based systems due to security and privacy concerns, these libraries primarily utilize the X11 client APIs. Therefore, to create a screenshot, the malware establishes a connection to the display server using the XOpenDisplay function and then requests a screenshot of the root window using XGetImage. In return, the display

PID	Process	Client ID	Event Mask	Captured Events
2698	key_ext	27	0x20	XI_RawKeyPress Events

Fig. 2. Xinputextensions detecting a keylogger that captures X11 events via input extensions.

server provides a buffer containing an XImage, representing a screenshot of the entire screen.

3.3.2. Memory forensic analysis

Both the screen capturing techniques utilizing XGetImage as well as the key-/mouseloggers relying on XQueryKeymap/-Pointer operate on the basis of a polling model, where the X11 server is periodically queried for information. This complicates their forensic analysis, as no registration for event notifications or analogous mechanisms is performed. Consequently, malicious activities cannot be detected by analyzing notification queues and event masks. To overcome this hurdle, we leverage the fact that malware querying the X11 server must use the XOpenDisplay function to establish a connection. On the server side, this reflects in initialization of a corresponding struct _Client instance.

We take use of this behavior to detect suspicious applications with the *xclients* Volatility plugin. The plugin first scans the X11 server process memory for _Client instances representing connected applications. Since this reveals both malicious and regular applications, we minimize false positives in the following way. We noted that every examined malware sample connects to the X11 server without generating a window. This is in contrast to nearly all regular applications, which connect to the X11 server for displaying a graphical user interface. Therefore, to detect malicious applications, we identify client applications that do not have a window associated. We do so by enumerating all parent-level windows maintained by the X11 server, and determining their owning clients by evaluating the _Window.drawable.id field. Then we map these client identifiers with those of the previously obtained clients instances contained in _Client.id. In case a client does not refer to any window, the plugin marks the corresponding applications as suspicious, allowing an investigator to identify malicious software (see Fig. 3).

3.4. Webcam & video recording

3.4.1. Malware implementation

In addition to capturing screenshots, some of the investigated malware samples can record streams of video devices. To do so, they utilize the V4L2 API, the standard interface to video devices in Linux environments. While malware usually tries to capture video footage of webcams, the V4L2 implementations are not limited to them and generalize to all available video devices. The V4L2 subystem of the Linux kernel offers access to every identified video device by providing a/dev/videoX character device. To capture streams of a video device, malware first obtains a file descriptor to the corresponding character device. It then creates a buffer of the type V4L2_BUF_TYPE_VIDEO_CAPTURE, and configures to map video data into its process memory by setting the V4L2_MEMORY_MMAP attribute. Finally, the malware queries the kernel to fill the buffer with video data by making an ioctl system call with VIDIOC_QUERYBUF (see Snippet 3).

```
const char *device = "/dev/video0";
int fd = open(device, O_RDWR);
/* ... */
memset(&buf, 0, sizeof(buf));
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;
ioctl(fd, VIDIOC_QUERYBUF, &buf)
```

Snippet 3: Webcam recording utilizing V4L.

PID	Process	Reason
813	key_query	Missing window ownership

Fig. 3. Xclients detecting a keylogger abusing XQueryKeymap.

PID	Process	FD	Path	Device	Inode	Type	Mode
1399	cam	3	/dev/video0	0:5	642	CHR	crw-rw----

Fig. 4. v4l2 plugin detecting capture of video devices.

3.4.2. Memory forensic analysis

As a prerequisite for capturing streams of video devices, the malware has to open a handle to the corresponding video device via the open system call. The Linux kernel names V4L2 devices according to the/dev/videoX scheme. We combine these facts to identify potential espionage attempts via open handles to V4L2 devices, and implement this detection approach in the *v4l2* Volatility plugin. The plugin leverages Linux kernel symbols to identify the kernel list that holds the control blocks of managed processes. Each process is represented by an instance of struct task_struct with a pointer to an array of open file descriptors in task_struct- > files- > fd_array. We use this array to iterate over all file handles of the process and resolve the corresponding file names. If a process opened a handle to a/dev/videoX character device, the plugin flags the process as potentially malicious, allowing an investigator to immediately identify video capturing software (see Fig. 4).

3.5. Microphone & audio recording

3.5.1. Malware implementation

Some of the examined malware samples can record audio streams, e.g., to spy on users by capturing microphone inputs. Recording microphone input requires access to the corresponding audio device. However, instead of directly accessing audio devices, Linux applications use the services of an audio server. To control audio servers, clients can utilize various shared libraries or directly use APIs from application frameworks such as Qt. This versatility also corresponds to the approaches implemented by the malware under investigation, we found malware utilizes the PulseAudio, the PortAudio, or the Qt client APIs to capture audio streams. However, regardless of which API is used, the capturing workflow is similar. We illustrate the workflow using the PulseAudio API (see Snippet 4). First, the pa_simple_new function establishes a connection to the audio server. By setting the PA_STREAM_RECORD flag, the server is set up for audio recording. Finally, the audio stream is requested from the server by calling pa_simple_read, which directly writes the audio stream to a byte buffer. At this point, the malware completes the recording by either exfiltrating the audio stream over the network or saving it to a file.

```
s = pa_simple_new(NULL, "PulseAudioRecorder",
↪  PA_STREAM_RECORD, NULL, "record", &ss, NULL,
↪  NULL, &error);
/* ... */
uint8_t buf[BUFSIZE];
pa_simple_read(s, buf, sizeof(buf), &error) < 0)
```

Snippet 4: Audio recording utilizing PulseAudio APIs.

3.5.2. Memory forensic analysis

While Linux distributions often shipped with the PulseAudio server in the past, now most distributions utilize a single audio server implementation, the modern PipeWire audio server (see Table 3), which facilitates the forensic analysis. The PipeWire server communicates with clients through a UNIX domain socket using the PipeWire native protocol. However, our evaluation (Table 3) shows that malware which

utilizes other APIs and network protocols can also successfully capture audio streams. This is due to the use of API-compatible daemons in the PipeWire project, which integrate with the PipeWire server and thus provide compatibility with other server implementations. Consequently, we focused on forensically analyzing the PipeWire server with the *pipewire* Volatility plugin to identify audio espionage attacks.

PipeWire internally manages connected clients as nodes, which are instances of the pw_node_impl structure. The structure exhibits a field pw_node_impl.info, which can be used to identify a list of properties pointed to by pw_node_info.prop. The prop field contains a pointer to a spa_dict structure, representing a dictionary of node properties that provide detailed information about the client and the requested resources. Therefore, the *pipewire* plugin scans the process memory of the PipeWire server instance to identify pw_node_impl instances. Following, it evaluates the node's properties, and identifies recording nodes using the media. class field. Thus, the plugin gives the investigators an immediate overview of processes that record audio streams (see Fig. 5).

4. Evaluation

With our work, we aim to contribute to the detection of espionage attacks and empower investigators with novel Linux forensic capabilities. To verify the applicability of the introduced methods in real-world scenarios, we performed multiple analyses of infected systems utilizing the previously introduced Volatility plugins.

4.1. Experimental setup

Considering that Linux distributions differ in their use of packages and software versions, we performed our evaluation across major Linux distributions to ensure the portability of the presented approaches. Therefore, we set up eight virtual machines and installed major Linux distributions with their default desktop environment and configuration. We denote the hard- and software used for virtualization in Table 4.

Since espionage attacks are not automatically triggered when the malware is launched but only when the control server requests them, we transferred the reverse-engineered espionage attacks to independently executable programs. This allows us to execute the same attack techniques without having to trigger the code paths in the malware. We deployed these attack programs to the virtual machines and mimicked user behavior by running desktop applications. We then iteratively executed an attack program, created a memory dump using AVML, and reverted the machine to a clean snapshot until all espionage attacks were executed on all distributions. Finally, we used the resulting memory dumps to validate the developed Volatility plugins and documented our results in Table 3.

4.2. Results

As can be seen from Table 3, the executed espionage techniques compromise all evaluated distributions. This even holds true for keylogging and screen capturing approaches on seemingly secure Wayland-based distributions, although only for non-native Wayland applications. The only exception is screen capturing on CentOS using the Qt framework, where Qt fails to access window surfaces due to unknown reasons. Overall, our results reveal a serious privacy threat, as major applications do not yet support Wayland, e.g., the Chromium browser or Electron-based applications. We discuss the reasons for this behavior in Sec. 5.

Notably, our evaluation shows that the Volatility plugins successfully detect all the espionage attacks that were conducted. The detection works across all evaluated Linux distributions, indicating the portability of our approaches. Finally, all distributions examined use PipeWire as an audio server, which underlines the practical applicability of detecting audio and microphone capture malware by analyzing the PipeWire process memory.

Table 3
Overview about espionage attack implementations, their impact on Linux distributions, and the Volatility plugins uncovering them.

Operating System				Espionage Technique										
				Key-/Mouse			Screen			Cam.	Micro.			
Distribution	**Desktop**	**Display Server**	**Audio Server**	XSelectInput	XiSelectEvents	XQueryKeymap	XGetImage	GDK	Qt	V4L2	PortAudio	PulseAudio	Qt
Ubuntu 24.04.	Gnome	Wayland/XWayland	PipeWire	◑	◑	◑	◑	◑	◑	●	●	●	●
Xubuntu 24.04.	Xfce	Xorg	PipeWire	●	●	●	●	●	●	●	●	●	●
Kubuntu 24.04.	KDE	Wayland/XWayland	PipeWire	◑	◑	◑	◑	◑	◑	●	●	●	●
Linux Mint 22.1	Cinnamon	Xorg	PipeWire	●	●	●	●	●	●	●	●	●	●
Fedora 41	Gnome	Wayland/XWayland	PipeWire	◑	◑	◑	◑	◑	◑	●	●	●	●
Debian 12.9	Gnome	Wayland/XWayland	PipeWire	◑	◑	◑	◑	◑	◑	●	●	●	●
CentOS 10	Gnome	Wayland/XWayland	PipeWire	◑	◑	◑	◑	◑	○	●	●	●	●
Rocky 9.5	Gnome	Wayland/XWayland	PipeWire	◑	◑	◑	◑	◑	◑	●	●	●	●
Volatility Plugins													
xevents				✓									
xinputextensions					✓								
xclients						✓	✓	✓	✓				
pipewire											✓	✓	✓
v4l2										✓			

● working on all applications.
◑ working on applications not natively supporting Wayland (e.g., Chrome, Electron apps, 1Password).

```
| PID  |  Process |          media.class |  media.name |       client.api |
| 1276 | mic_port | Steam/Input/Audio | ALSA Capture | pipewire-pulse |
```

Fig. 5. The pipewire plugin detecting capture of microphone input.

Table 4
Experimental setup.

Hardware		Software	
CPU	2 vCPU	Hypervisor	QEMU 9.2.0
RAM	1GiB	Mem. Acquisition	AVML 0.14.0
Disk Size	32GiB	Mem. Analysis	Volatility 3 2.11

5. Discussion

This study analyzed spy techniques utilized by ten different malware families. Our investigation revealed implementations of various espionage techniques, utilizing multiple shared libraries and programming languages, which demonstrates that Linux desktops are tangible targets for espionage operations. Our analysis showed that the samples investigated primarily targeted Linux desktop services, so currently available memory forensic techniques may fall short in detecting them. We therefore introduced novel memory forensic approaches for analyzing core services of the Linux desktop ecosystem, and our proof-of-concept implementations proved effective in detecting espionage attacks across all evaluated desktop distributions.

5.1. X11, Wayland and XWayland

Our analysis indicates that, despite the emergence of Wayland as a successor to X11, the latter remains the primary target for malware developers. A potential reason for this is the greater attack surface offered by X11 since espionage attempts can be executed with regular user permissions. While Wayland-based desktop environments should mitigate these espionage techniques, our evaluation shows that all evaluated desktop distributions use XWayland for backward compatibility reasons. XWayland provides X-protocol services to non-native Wayland applications, which allows them to run non-ported software but also retains their susceptibility to X11-based espionage techniques. This poses a serious privacy threat, impacting popular applications like Chromium and Electron-based software, and therefore messaging platforms such as Slack and Discord, as well as the password manager 1Password.

5.2. Limitations

Research has shown that memory forensics can be affected by page smearing (Pagani et al., 2019; Ottmann et al., 2023) or virtual memory swapping, e.g., in case of scarcity of resources due to high workloads. To provide a realistic evaluation, we activated the use of swap files and emulated typical user workload. While memory forensics of userland processes can be particularly susceptible to memory swapping, we have not encountered a situation where the introduced approaches failed to correctly analyze an acquired dump. Notably, despite the fact that only a small amount of virtual memory was assigned to the virtual machines. However, high-intensive system workloads could negatively influence the acquisition of memory dumps, and therefore influence analysis results. In future work, a measurement with high-intensive workloads can be performed to evaluate the impact on the presented approaches.

5.3. Future work

In some cases, malware implementations use plugin or extension architectures, so that malware samples of the same family may exhibit different functionality. Therefore, our analysis may not cover all potentially available modules and techniques, but only those included in the malware samples we were able to obtain. For instance, the FinFisher malware has been reported to include an audio recording module (Amnesty International, 2020), which was not included in our samples. While we built a malware dataset using publicly available information, we expect commercial malware feeds, such as VirusTotal, to hold even more samples and malware families targeting the Linux desktop. As the authors do not have access to these commercial feeds, their evaluation is left for future work.

During our study, we found that desktop environments, such as Gnome and KDE, provide functionality that is unavailable in Wayland-based systems due to the enhanced security model. This applies in particular to screen capturing functionality, which is often demanded by users, but not natively supported in Wayland. For instance, we found that both Gnome and KDE allow applications to request screen recordings and screenshots via D-Bus services, and some Wayland compositors offer similar functionality via portal extensions. Future work should explore, in which way these specific implementations may pose a privacy risk, enhancing the understanding of potentially evolving threats.

6. Related work

Even before the reported rise of Linux malware (Cozzi et al., 2018; Ramamoorthy et al., 2024; Chierzi and Mercês, 2021), memory forensics has been a critical component of forensic analyses investigating Linux systems (Ligh et al., 2014; Maxwell et al., 2013), e.g., by evaluating kernel structures to identify hidden processes or network connections that indicate rootkit presence.

In recent years, Linux memory forensics has seen significant advancements, e.g., by the adoption of userland heap analyses (Block and Dewald, 2017) and novel approaches that provide hardware and kernel-agnostic memory evaluation (Oliveri and Balzarotti, 2022; Oliveri et al., 2023), aiming to make Linux memory analyses a scalable, cross-architecture forensic approach. Thereby, its wide adoption is mirrored by various forensic applications, e.g., to recover messages and passwords from instant messaging applications (Davis et al., 2022) or network attack (Zhang et al., 2022) and eBPF rootkit detection (Zaidenberg et al., 2025).

Recently, related work called for increasing efforts on memory analyses of userland subsystems for detecting sophisticated malware (Case and Richard, 2017), as happened with Objective C (Case and Richard, 2016), Swift (Manna et al., 2021) and. NET runtimes (Manna et al., 2022). In contrast, memory analyses of GUI environments have primarily been used to extract user data, e.g., screenshots (Saltaformaggio et al., 2015) or documents (Saltaformaggio et al., 2014). In line with these efforts, open source Volatility plugins to extract window attributes and screenshots from Linux-based X11 servers were published (Geek, 2023; eurecom-s3/linux_screenshot_xwindows and original, 2024). However, the potential for memory forensics focusing on malware detection in the Linux desktop ecosystem remains largely unexplored. As a result, current memory forensic approaches can reveal malware targeting Linux kernel functionality (Ligh et al., 2014; Case et al., 2022), but do not cover espionage attacks that abuse Linux desktop functionality.

7. Conclusion

In this work, we investigated the espionage capabilities of Linux desktop malware and introduced novel methods to detect them. Our analysis reveals implementations of sophisticated spying capabilities, which pose a significant threat to user privacy. Since the investigated malware is used by state actors and APT groups known to hide their traces, we proposed memory forensic techniques to uncover espionage attempts. Our evaluation shows that the introduced methods successfully detect espionage attacks and work across major Linux distributions. Thus, this study provides insights into a previously understudied

threat and offers investigators practical tools for responding to it.

Acknowledgement

This research work was supported by the National Research Center for Applied Cybersecurity ATHENE.

References

Amnesty International, 2020. German-made FinSpy spyware found in Egypt, and Mac and Linux versions revealed. https://www.amnesty.org/en/latest/research/2020/09/german-made-\allowbreakfinspy-spyware-found-in-egypt-and\allowbreak-mac-and-linux-versions-revealed/.

Block, F., Dewald, A., 2017. Linux memory forensics: Dissecting the user space process heap. Digit. Invest. 22, S66–S75. https://doi.org/10.1016/j.diin.2017.06.002. URL. https://www.sciencedirect.com/science/article/pii/S1742287617301895.

Case, A., Richard, G.G., 2016. Detecting objective-C malware through memory forensics. Digit. Invest. 18, S3–S10. https://doi.org/10.1016/j.diin.2016.04.017. URL. https://www.sciencedirect.com/science/article/pii/S1742287616300524.

Case, A., Richard, G.G., 2017. Memory forensics: the path forward. Digit. Invest. 20, 23–33. https://doi.org/10.1016/j.diin.2016.12.004. URL. https://linkinghub.elsevier.com/retrieve/pii/S1742287616301529.

Case, A., Moreira, G., Sellers, A., 2022. New Memory Forensics Techniques to Defeat Device Monitoring Malware.

Chierzi, V., Mercês, F., 2021. Evolution of IoT linux malware: a MITRE ATT&CK TTP based approach. In: 2021 APWG Symposium on Electronic Crime Research (eCrime), pp. 1–11. https://doi.org/10.1109/eCrime54498.2021.9738756 iSSN: 2159-1245. https://ieeexplore.ieee.org/abstract/document/9738756/citations?tabFilter=papers#citations.

Cozzi, E., Graziano, M., Fratantonio, Y., Balzarotti, D., 2018. Understanding linux malware. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 161–175. https://doi.org/10.1109/SP.2018.00054 iSSN: 2375-1207. https://ieeexplore.ieee.org/abstract/document/8418602.

Cyble, 2024. Cryptocurrency lures and pupy RAT: analysing the UTG-Q-010 campaign - Cyble, section: APT. https://cyble.com/blog/analysing-the-utg-q-010-campaign/.

Davis, M., McInnes, B., Ahmed, I., 2022. Forensic investigation of instant messaging services on linux OS: Discord and Slack as case studies. Forensic Sci. Int.: Digit. Invest. 42, 301401. https://doi.org/10.1016/j.fsidi.2022.301401. URL. https://www.sciencedirect.com/science/article/pii/S2666281722000828.

eurecom-s3/linux_screenshot_xwindows, original-date: 2018-05-15T11:59:43Z. https://github.com/eurecom-s3/linux_screenshot_xwindows, 2024.

Geek, B.t., 2023. bridgeythegeek/linux_xwindows, Original-Date: 2017-12-03T11:03:52Z. https://github.com/bridgeythegeek/linux_xwindows.

GitHub, I., 2025. Github. https://github.com. (Accessed 29 January 2025).

Hillenius, G., 2013. 'Open source only' at Dutch police Internet forensics. https://interoperable-europe.ec.europa.eu/collection/open-source-observatory-osor/news/open-source-only-dutch-p.

Kaspersky, 2020. An overview of targeted attacks and APTs on Linux. https://securelist.com/an-overview-of-targeted-attacks-and-apts-on-linux/98440/.

Ligh, M.H., Case, A., Levy, J., Walters, A., 2014. The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. John Wiley & Sons.

linux.com, 2007. Open technology within DoD, section: news. https://www.linux.com/news/open-technology-within-dod-intel-systems/.

MalwareBazaar, 2025. Malwarebazaar database. https://bazaar.abuse.ch. (Accessed 29 January 2025).

Manna, M., Case, A., Ali-Gombe, A., Richard, G.G., 2021. Modern macOS userland runtime analysis. Forensic Sci. Int.: Digit. Invest. 38, 301221. https://doi.org/10.1016/j.fsidi.2021.301221. URL. https://www.sciencedirect.com/science/article/pii/S2666281721001293.

Manna, M., Case, A., Ali-Gombe, A., Richard, G.G., 2022. Memory analysis of .NET and .net core applications. Forensic Sci. Int.: Digit. Invest. 42, 301404. https://doi.org/10.1016/j.fsidi.2022.301404. URL. https://www.sciencedirect.com/science/article/pii/S2666281722000853.

Maxwell, R., Malin, C.H., Casey, E., Aquilina, J.M., 2013. In: Malware Forensics Field Guide for Linux Systems: Digital Forensics Field Guide, illustrated edition Edition. Syngress Media, Amsterdam Boston.

Muenster, F., 2025. Uncovering linux desktop espionage. https://github.com/FHMS-ITS/uncovering-linux-desktop-espionage/. (Accessed 29 January 2025).

Oliveri, A., Balzarotti, D., 2022. In the land of MMUs: multiarchitecture OS-agnostic virtual memory forensics. ACM Transactions on Privacy and Security 25 (4), 1–32. https://doi.org/10.1145/3528102. URL. https://dl.acm.org/doi/10.1145/3528102.

Oliveri, A., Dell'Amico, M., Balzarotti, D., 2023. An OS-agnostic approach to memory forensics. In: Proceedings 2023 Network and Distributed System Security Symposium. Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2023.23398. URL. https://www.ndss-symposium.org/wp-content/uploads/2023/02/ndss2023_s398_paper.pdf.

Ottmann, J., Breitinger, F., Freiling, F., 2023. An experimental assessment of inconsistencies in memory forensics. ACM Trans. Priv. Secur. 27 (1). https://doi.org/10.1145/3628600.

Pagani, F., Fedorov, O., Balzarotti, D., 2019. Introducing the temporal dimension to memory forensics. ACM Transactions on Privacy and Security 22 (2), 1–21, publisher: Association for Computing Machinery. https://orcid.org/0000-0002-4357-9804. https://dl.acm.org/doi/abs/10.1145/3310355.

Peri, D., 2023. India Defence Ministry to Replace Microsoft OS with Maya. The Hindu. URL. https://www.thehindu.com/news/national/defence-ministry-to-replace-microsoft-os-with-maya/article67172875.ece.

Ramamoorthy, J., Varol, C., Shashidhar, N., 2024. APT Warfare: Technical Arsenal and Target Profiles of Linux Malware in Advanced Persistent Threats.

Saltaformaggio, B., Gu, Z., Zhang, X., Xu, D., 2014. DSCRETE: Automatic Rendering of Forensic Information from Memory Images via Application Logic Reuse. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/saltaformaggio.

Saltaformaggio, B., Bhatia, R., Gu, Z., Zhang, X., Xu, D., 2015. GUITAR: piecing together android app GUIs from memory images. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, Denver Colorado USA, pp. 120–132. https://doi.org/10.1145/2810103.2813650. URL. https://dl.acm.org/doi/10.1145/2810103.2813650.

Steam, 2024. Steam hardware & software survey. https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam?platform=combined.

Talos, C., 2023. Lazarus Group's infrastructure reuse leads to discovery of new malware. https://blog.talosintelligence.com/lazarus-collectionrat/.

Underground, V., 2025. Vx underground. https://vx-underground.org. (Accessed 29 January 2025).

S. Vaughan-Nichols, 5 reasons why desktop Linux is finally growing in popularity. URL https://www.zdnet.com/article/5-reasons-why-desktop-\allowbreaklinux-is-finally-growing-in-popularity/.

Volatility Foundation, 2025. Volatility 3.0. https://github.com/volatilityfoundation/volatility3.

Zaidenberg, N., Kiperberg, M., Menachi, E., Eitani, A., 2025. Detecting eBPF Rootkits Using Virtualization and Memory Forensics, pp. 254–261. URL. https://www.scitepress.org/Link.aspx?doi=10.5220/0012470800003648.

Zhang, Z., Liu, Z., Bai, J., 2022. Network attack detection model based on Linux memory forensics. In: 2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), pp. 931–935. https://doi.org/10.1109/ICMTMA54903.2022.00189 iSSN: 2157-1481. https://ieeexplore.ieee.org/abstract/document/9724000.