Linux Offensive – State of the Art

Zacharia Mansouri

CYBER DEFENCE LAB

October 28, 2026

Linux Offensive – State of the Art
Zacharia Mansouri
October 28, 2026

Cyber Defence Lab
https://cylab.be

BibTeX citation:

```
@techreport{citekey,
title = {Linux Offensive - State of the Art},
author = {Zacharia Mansouri},
institution = {Cyber Defence Lab},
year = {2026},
month = {10},
}
```

This work is licensed under a Creative Commons "Attribution 4.0 International" license.

Transparency Regarding AI Usage

Artificial Intelligence was used in this report to identify data within texts and to format the content according to academic standards.

Abstract

As Linux remains the keystone of modern cloud and IoT ecosystems, threat actors are deploying increasingly sophisticated malware. This report first examines contemporary Linux execution environments, container risks, and emerging exploitation vectors like eBPF. It then comprehensively maps these threats across the 18-phase Unified Kill Chain, revealing how modern, highly modular malware collapses traditional kill chain boundaries to simultaneously achieve persistence, evasion, and lateral movement.

Keywords: Linux Security, Linux Malware, Unified Kill Chain, Threat Intelligence, eBPF Exploitation, Rootkits, Container Security

Contents

1 General Overview **1**
- 1.1 Web-based Intrusions . . . 1
- 1.2 Execution Environments . . . 2
  - 1.2.1 Computer Architectures . . . 3
  - 1.2.2 Loaders and Libraries . . . 3
  - 1.2.3 Operating Systems . . . 4
  - 1.2.4 Containers . . . 5
- 1.3 Malware Families . . . 6
- 1.4 Advanced Exploitation . . . 7
  - 1.4.1 eBPF . . . 8
  - 1.4.2 Command & Control Innovations . . . 9

2 Linux Malware – Unified Kill Chain Mapping **11**
- 2.1 Reconnaissance . . . 12
  - 2.1.1 Active Reconnaissance . . . 12
  - 2.1.2 Passive Reconnaissance . . . 13
- 2.2 Resource Development . . . 14
  - 2.2.1 Malware Development and Adaptation . . . 14
  - 2.2.2 Infrastructure Setup and Staging . . . 14
  - 2.2.3 Evasion and Persistence Features . . . 15
  - 2.2.4 Tooling . . . 15
- 2.3 Delivery . . . 16
  - 2.3.1 Phishing and Social Engineering . . . 16
  - 2.3.2 Network Delivery to Exposed Services . . . 16
  - 2.3.3 Supply Chain and Trojanized Software . . . 17
  - 2.3.4 Staged Delivery Mechanisms . . . 17
- 2.4 Social Engineering . . . 18
  - 2.4.1 ClickFix-style Social Engineering . . . 18
  - 2.4.2 Phishing and Spear Phishing . . . 18
- 2.5 Exploitation . . . 20
  - 2.5.1 Semantic Vulnerabilities in Linux Kernel Security . . . 20
  - 2.5.2 Exploiting Public-Facing Applications and Services . . . 21
  - 2.5.3 Local Privilege Escalation Exploits . . . 21

2.6 Persistence . . . 23
2.6.1 Bootloader and Init System Manipulation . . . 23
2.6.2 Scheduled Tasks and Automation . . . 23
2.6.3 System Process Modification . . . 24
2.6.4 File System Manipulation and Infection . . . 24
2.7 Defense Evasion . . . 26
2.7.1 Code and Binary Manipulation . . . 26
2.7.2 Obstructing Code Flow and Analysis . . . 27
2.7.3 Evasion and Anti-Debugging . . . 27
2.7.4 System Configuration and Process Hiding . . . 28
2.7.5 Network Evasion . . . 29
2.7.6 Kernel-Level Evasion (eBPF) . . . 30
2.8 Command & Control . . . 31
2.8.1 Establishing Communication Channels . . . 31
2.8.2 Obfuscation and Evasion in Communication . . . 31
2.8.3 Command Execution and Capabilities . . . 32
2.8.4 Internal Communication . . . 32
2.9 Pivoting . . . 34
2.10 Discovery . . . 35
2.10.1 System and Host Information Gathering . . . 35
2.10.2 Process and Service Enumeration . . . 35
2.10.3 Network Reconnaissance . . . 36
2.10.4 Credential and File System Discovery . . . 36
2.10.5 Anti-Analysis-Related Discovery . . . 37
2.11 Privilege Escalation . . . 38
2.11.1 Exploiting Known Vulnerabilities . . . 38
2.11.2 Abusing System Mechanisms . . . 38
2.11.3 Credential and Process Manipulation . . . 39
2.11.4 Targeting Virtualization Platforms . . . 39
2.12 Execution . . . 41
2.12.1 Command and Scripting Interpreter Abuse . . . 41
2.12.2 File-Based Execution . . . 41
2.12.3 Process Manipulation and Injection . . . 42
2.13 Credential Access . . . 43
2.13.1 SSH Credential Theft and Manipulation . . . 43
2.13.2 System File Manipulation and Authentication Bypass . . . 43
2.13.3 Deployment of Stealers & Network Sniffing . . . 44
2.14 Lateral Movement . . . 45
2.14.1 Credential Reuse and Stolen Credentials . . . 45
2.14.2 Abusing Remote Services . . . 45
2.14.3 Pivoting and Tunneling . . . 46

2.14.4 Execution of Commands and Payload Deployment 46
2.14.5 AitM Attacks 46
2.15 Collection 47
2.15.1 Data Staging and Archiving 47
2.15.2 Automated Input Capture 47
2.16 Exfiltration 48
2.16.1 Exfiltration via C2 Channels 48
2.16.2 Abuse of Legitimate Cloud and Web Utilities 48
2.16.3 Covert and Obfuscated Channels 48
2.17 Impact 50
2.17.1 Data and System Destruction 50
2.17.2 Denial of Service 50
2.17.3 Financial and Data Manipulation 51
2.17.4 Resource Hijacking 51
2.18 Objectives 52

3 Conclusion 53

1. General Overview

For many years, the antivirus industry treated *Linux* malware as a minor concern. However, since late 2014, the increasing and diversifying threat landscape of *Linux* malware started to be recognized by *VirusTotal* as a growing concern for the security community [1].

Linux is no longer a niche Operating System (OS). As its presence across web services and execution environments grew and still expands, it has become a target for various malware families.

This first chapter examines how Linux threats are structured. Specifically, Sec. 1.1 details the web-based intrusion vectors and vulnerability trends exploited for initial access. Sec. 1.2 analyzes the diverse hardware architectures, loaders, libraries and containerized microservices that constitute the modern runtime target surface. Sec. 1.3 categorizes the dominant malware families and botnet prevalences across the ecosystem, while Sec. 1.4 explores modern post-exploitation techniques, focusing on kernel-level extended Berkeley Packet Filter (eBPF) abuse and evolving Command & Control (C2) infrastructures.

1.1. Web-based Intrusions

As of late 2025, the *Linux* OS still acts as the backbone of global digital infrastructure, powering 58% of identifiable websites, which grew up to reach 62% around mid-2026 [2]. Its prevalence exists across major cloud environments such as *Amazon Web Services*, *Google Cloud*, *Microsoft Azure* and *OpenStack* due to the wide adoption of cloud-native technologies such as *Kubernetes* [3].

Web-based attacks frequently rely on exploiting specific, well-known vulnerabilities. The most exploited *Linux* vulnerabilities observed in 2022 showed in Table 1.1 were heavily oriented towards web infrastructure flaws [4]. While the threat landscape rapidly evolves, some vulnerabilities exploited for initial access remain the same. A look at 2022 data shows that old web security flaws (such as *Log4j* and *Apache Struts*) created the original attack methods that hackers still heavily rely on in 2026 [5], [6].

A significant proportion of web attacks use flaws listed in the TOP 10 Open World-

Vulnerability	Presence
CVE-2021-44228 (Log4j vulnerability)	32.5%
Apache Struts (CVE-2017-12611 and CVE-2018-11776)	19.5%
OpenSSH vulnerability (CVE-2018-15473)	19.4%
Others	28.6%

Table 1.1. Most Exploited *Linux* Vulnerabilities in 2022

wide Application Security Project (OWASP) attacks list, accounting for 70.5% of the total attacks in 2022 [4]. The most common attack vectors seen within the OWASP category are listed in Table 1.2.

Attack	Presence
Cross-site scripting (XSS)	32.6%
SQL injection	24.3%
Directory Traversal	23.1%
Command injection	16.9%
Insecure deserialization	1.9%
XML External Entity (XEE)	1.2%

Table 1.2. Most Common OWASP Attacks in 2022

These exploits are directly related to the deployment of persistent malware. For example, successful exploitation of web application vulnerabilities like *SQL* injection, file inclusion or command injection is the initial mechanism used by threat actors to install malicious scripts as web shells. Once installed, these provide a powerful interface for controlling the compromised server and its network [4]. To understand how malware establishes execution and persists within a target host, we must examine the diverse hardware architectures, binary loaders, OS Application Binary Interface (ABI) and containerized microservices that compose the modern runtime environment.

1.2. Execution Environments

Linux is deployed across a multitude of architectures (e.g., *ARM*, *MIPS*, *x86*), powering diverse ecosystems that range from embedded systems and Internet of Things (IoT) devices to cloud, desktop and mobile environments. Executables operating within these environments can be linked either statically or dynamically, rely on varying library dependencies (e.g., *glibc*, *musl*) and must manage compatibility constraints related to the target kernel ABIs. This immense versatility creates

a complex execution landscape that threat actors actively exploit through techniques such as file header corruption, runtime execution anomalies and analysis tool evasion. On top of that, the widespread adoption of containerized microservices introduces an additional layer of execution environments, bringing container base image vulnerabilities into the overall attack surface.

1.2.1. Computer Architectures

The *Linux* OS targets a highly diverse range of environments, including personal computers, servers, mobile devices (e.g., *Android*), embedded systems and cloud environments [1], [4]. This includes various devices such as internet routers, printers, surveillance cameras and smart TVs. We can also note the use of *Linux*-based solutions for safety-critical systems in medical devices, autonomous vehicles and spacecraft [7]. That diversity of architectures complicates malware targeting, but embedded systems and IoT devices remain easy targets due to vulnerabilities coming from the pressure to quickly bring new products to the market. For example, the famous *Mirai* botnet that was active between 2016 and 2017 primarily targeted embedded systems and IoT devices for large-scale Distributed Denial of Service (DDoS) attacks and supports at least three architectures (*MIPS 32-bit*, *ARM 32-bit* and *x86 32-bit*) [8]. Certain families, such as *ARM*, are particularly challenging to support due to the large number of different *CPU* architectures they contain [1]. Alongside the hardware constraints imposed by diverse physical architectures, deploying and executing code across these disparate environments requires managing complex software loading mechanisms, binary formats and library dependencies.

1.2.2. Loaders and Libraries

As detailed by Cozzi et al. [1], loaders are used for loading an executable but also setting it up in memory and initiating its execution. The Executable and Linkable Format (ELF) headers contain information to properly load and execute a binary file. Additionally, while some *Linux* programs are dynamically linked with alternative libraries such as `uClibc` or `musl`, which are smaller and more efficient compared to the traditional `glibc`, malware authors frequently use static linking to overcome the challenges of diverse target environments and missing dependencies. This method increases binary portability, allowing execution even without installed dependencies, while simultaneously complicating reverse engineering by making it difficult to identify the library functions that are called. However, because statically linked applications do not rely on an external wrapper for system calls, they

remain vulnerable to runtime crashes if the target system's kernel ABI differs from expectations, such as when executing across varying OS like *Linux* and *FreeBSD*.

1.2.3. Operating Systems

In the same study, Cozzi et al. [1] observe that the target OS can be specified in ELF headers within an `OS/ABI` field, but that field is often unreliable in practice. Indeed, current *Linux* kernels generally ignore the `OS/ABI` field. A binary specifying *FreeBSD* as its `OS/ABI` value can still be a valid *Linux* program and be properly loaded and executed under *Linux*, provided it is dynamically linked. Exploiting this behavior, malware developers may intentionally tamper with the ELF headers, for instance, reporting a false OS ABI (e.g., *FreeBSD*) while the program is actually compiled for *Linux*, as a technique to deceive analysts. Another common anomalous technique is removing all information about the ELF sections. Other techniques involve deliberately malformed or corrupted header or section information, intended to confuse or halt analysis tools. For example, malformed or corrupted section header information can result from invalid values in fields such as `e_shoff` (offset of the section header table), `e_shnum` (number of entries) or `e_shentsize` (size of section entries) in the ELF header. Invalid ELF files can be created by having the segment header table or the section header table point beyond the file data. Other techniques include overlapping the ELF header and segment or using a wrong string table index (`e_shstrndx`).

The success of ELF header manipulation is evident in how various common analysis tools react to corrupted information [1]:

- *readelf* complains about the absence of certain fields.
- *pyelftools* denies further analysis if the section header table is corrupted, though it can still parse ELF files if the table is explicitly declared as empty.
- GNU Debugger (GDB) shows a severe lack of resilience when encountering corrupted information from a malformed section header table. The presence of an invalid value can result in GDB being unable to recognize the ELF binary, thus preventing it from starting the program.

In contrast, *IDA Pro 7* is noted for being able to correctly handle the presence of any corrupted section information or other fields that do not affect the program's execution [1].

1.2.4. Containers

Beyond OSs and binary structures, as detailed by Trend Micro [4], modern software deployment increasingly relies on containerized microservices. Container technology, particularly *Docker* and its *Linux*-based containers, has become a fundamental component of modern software deployment, serving as a staple in *DevOps* for powering microservice architectures and cloud-native applications. Containers provide a lightweight, flexible and scalable way to package and deploy applications. These containers are built from images that bundle an application alongside its dependencies. *Docker*'s public registry, *DockerHub*, hosts millions of such images. The security posture of containers is directly linked to the security of these underlying images. The most popular official images on *DockerHub* often include base OSs (such as *Ubuntu*, *Alpine* and *CentOS*) and typical application environments (like *Node.js*, *Python* and *MySQL*).

The same report notes that a significant security issue is the presence of vulnerabilities within images [4]. Many images on *DockerHub* contain known vulnerabilities, frequently because they are built upon outdated base images. Other common issues that increase the attack surface include the continued use of outdated or vulnerable software within the image layers and the presence of tools or libraries that are unnecessary for the application to run, such as unneeded compilers, debugging tools or documentation files.

Furthermore, Trend Micro's scan of the top 20 most downloaded official images from *DockerHub* revealed that a large number of vulnerabilities are consistently present in key base images [4]. Table 1.3 details the 10 most vulnerable images based on the count of identified vulnerabilities across various severity levels.

Image	Critical	High	Medium	Low	Total
python	18	230	223	65	536
node	17	223	218	65	523
golang	10	98	65	11	184
wordpress	8	87	83	16	194
nginx	5	26	13	8	52
httpd	3	23	11	7	44
redis	3	15	4	9	31
mysql	2	18	13	1	34
postgres	1	24	8	8	41
memcached	1	14	2	7	24

Table 1.3. 10 Most Vulnerable Official Images from *DockerHub* (Top 20 Downloaded, 2022) [4]

1.3. Malware Families

While execution environments and binaries define how malware operates on a system, the threat landscape itself is populated by distinct behavioral groups and capability classes, ranging from widespread botnets to stealthy rootkits.

Botnets, often dedicated to running DDoS attacks, dominate the *Linux*-based malware landscape, accounting for 69% of samples across more than 25 families [1], [4]. Attackers frequently harvest poorly protected IoT devices to incorporate them into large, remotely controlled botnets [1].

As shown in Table 1.4, in 2022, specific types of malware showed the following prevalence in the *Linux* ecosystem [4].

Malware Family	**Presence**
Webshells	49.6%
Trojan	29.4%
Backdoor	11.6%
Cryptocurrency Miner	4.2%
Adware	2.9%
Phishing	1.8%
Hacktool	0.2%
Other	0.3%

Table 1.4. Most Common Encountered Linux Malware Families in 2022 [4]

As further detailed by Trend Micro [4], these families exhibit distinct operational characteristics:

- Web shells: these malicious scripts are installed on a server after threat actors successfully exploit web application vulnerabilities, such as *SQL* injection, file inclusion or command injection flaws. Once installed, web shells provide a powerful interface for controlling the compromised server and network.
- Trojan: malware such as *Trojan.Siggen28.58279* function as Remote Access Trojans (RATs) capable of infecting both *Windows* and *Linux* machines [9].
- Cryptocurrency Miners: these attacks exploit misconfigurations and gaps, particularly in cloud environments using public-facing *Linux*-based applications, to hijack the target's computing resources for mining cryptocurrencies.
- Ransomware: these attacks are increasing on *Linux* systems, often exploiting poor system configurations, outdated software or successful phishing at-

tempts. Ransomware attacks aim to encrypt and/or steal files and data from the victim in order to extort a ransom.

- Rootkits: these are stealthy forms of malware that typically necessitate kernel-level access to operate, so they often exploit vulnerabilities that allow privilege escalation. Once installed, a rootkit can manipulate system calls and logs to cloak its activity, making detection and removal difficult. Advanced Persistent Threat (APT) groups have demonstrated the abuse of Berkeley Packet Filter (BPF) filters for the installation of backdoors, which presents significant obstacles in detection, blocking and analysis due to the nature of its abuse [9].

After establishing initial access vectors and outlining behavioral threat profiles, advanced adversary campaigns use specialized post-exploitation tools, kernel-level capabilities and innovative communication channels to sustain long-term operational control.

1.4. Advanced Exploitation

As highlighted by *Doctor Web* [9], the post-exploitation phase of advanced attacks, following the establishment of initial access and concealment mechanisms like rootkits, involves the deployment of specialized tools, often RATs. There is a noticeable trend among threat actors to move away from widely recognized commercial frameworks toward custom and open-source alternatives for these post-exploitation activities.

The same analysis notes that historically, proprietary tools like *Cobalt Strike* and *Metasploit* were highly popular choices for installing RATs and carrying out post-exploitation tasks [9]. Following the widespread distribution of pirated versions of Cobalt Strike in 2022, its usage in cyberattacks grew significantly. In response, the manufacturer began attempting to track and block unofficial servers using the proprietary tool. For that reason, attackers are increasingly turning to open-source frameworks for their post-exploitation operations, attracting less attention from security researchers and defenders.

To drive these campaigns, attackers increasingly use low-level kernel capabilities and resilient network channels, specifically through eBPF feature abuse and modern C2 innovations.

1.4.1. eBPF

As stated in [10], the extended Berkeley Packet Filter (eBPF) is a kernel technology that allows to implement kernel code features such as performance tracing of pretty much any aspect of a system, high-performance networking with built-in visibility, detection of malicious activity, prevention and other related capabilities. The eBPF technology builds upon the concepts introduced in 1993 inside the Berkeley Software Distribution (BSD) packet filter paper [11], which describes a pseudo-machine designed to execute programs (filters) that assess network packets to decide whether they should be accepted or rejected and it was introduced to the *Linux* kernel version `2.1.75` in 1997. These filters were originally written using the BPF instruction set, a 32-bit architecture that closely mirrors assembly language. BPF was first integrated into *Linux* for use in the `tcpdump` [12] utility, providing an efficient mechanism for capturing packets intended for tracing. Introduced in the *Linux* kernel version `3.18`, eBPF brought major improvements including: a redesigned BPF instruction set optimized for 64-bit systems, the `bpf()` system call for user-space interaction with eBPF, maps for data sharing between programs and user-space, new BPF helper functions to extend capabilities and an eBPF verifier ensuring program safety [10].

According to the *MITRE* vulnerabilities list from Fig. 1.1 related to the `bpf` keyword [13], a total of 548 BPF vulnerabilities has been discovered as of September 7, 2026. A sharp spike occurred in 2024, where 126 BPF vulnerabilities were discovered in that year alone, exceeded in 2026 with currently 165 discovered vulnerabilities for this still ongoing 2026 year. Despite accumulating a smaller number of 104 vulnerabilities, the `ebpf` keyword [14] shows a growth that highlights the emerging significance of eBPF as a target for exploitation within the *Linux* kernel. Indeed, in 2026, 35 vulnerabilities have been exposed for now, which is already higher than for any previous year and seems to join the growing tendencies of BPF.

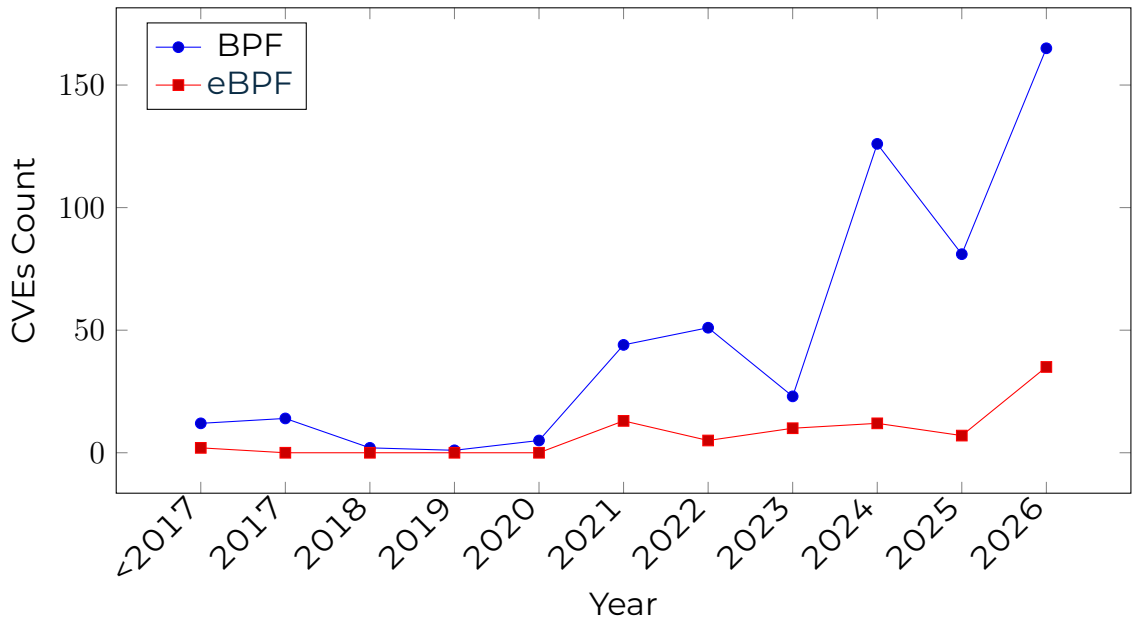


Figure 1.1. Number of Common Vulnerabilities and Exposures (CVEs) related to BPF and eBPF up to September 2026

Apart from direct exploitation, threat actors and APT groups abuse legitimate eBPF features primarily for defense evasion and the installation of backdoors [4]. As eBPF programs execute directly within the kernel space and interact dynamically with system events [10], they can be weaponized to manipulate system calls at a low level (e.g., altering the output of monitoring utilities). This capability makes the abuse inherently difficult to investigate, presenting security teams with novel obstacles in detection, blocking and analysis [4]. In some observed hacking campaigns, some eBPF rootkits are parts of a multiple stage compromise [9]:

1. Initial Concealment: attackers deploy a modified version of the `pidhide` program from the `bad-bpf`[1] project to establish system stealth.
2. Secondary Payload Installation: this eBPF-based concealment masks the installation of a traditional Loadable Kernel Module (LKM) rootkit, loaded via the `insmod` command.
3. Deep Evasion: once active, the LKM rootkit suppresses system visibility by hiding all files and folders prefixed with a specific string (e.g., `aclsmos_`).

1.4.2. Command & Control Innovations

APT groups have adopted innovative C2 infrastructure to evade network detection and ensure persistent communication with compromised *Linux* machines [9].

[1] https://github.com/pathtofile/bad-bpf

As further explained by *Doctor Web* [9], previous attacks were relying on services like *Dropbox*, *Google Drive*, *OneDrive* and *Discord* for C2 purposes. In modern attacks, the malware configuration data, which includes C2 information, is often stored in an encrypted form on public, trusted sites. This technique is referred to as serverless C2 or cloud-based C2. Configuration is stored on public sites such as *GitHub*/*GitLab* and even blog pages. The malware interacts with these servers, including their Content Delivery Networks (CDNs), to retrieve the necessary files. The configuration data retrieved from these sites is stored in an encrypted format, typically using a combination of a private key and *XOR* encryption. Using public platforms for C2 infrastructure helps the associated network traffic draw less attention compared to traditional, dedicated C2 servers.

2. Linux Malware – Unified Kill Chain Mapping

Having established the modern landscape of *Linux* threats, it is important to understand how these elements are orchestrated during a live intrusion. To illustrate how threat actors chain them together during an active campaign, the following chapter maps these technical mechanisms against a chronological attack model.

The Unified Kill Chain (UKC) is designed to detail the tactics used in modern cyber attacks by APTs and ransomware groups. The UKC is divided into 18 phases to fully describe attacks. It synthesizes the structure of the Cyber Kill Chain (CKC)[1] with the tactical analysis of the *MITRE ATT&CK*[2] framework to produce an end-to-end model of intrusion campaigns. It extends both frameworks by incorporating additional phases such as pivoting or impact, the latter having been included later into *MITRE ATT&CK*. The framework was developed through case studies conducted on transparent cyber attacks executed by *Fox-IT*[3]'s red team and applied to intrusions attributed to the *APT28* group[4]. These analyses enabled evaluation and refinement of the framework [15].

Where *MITRE ATT&CK* is an encyclopedic matrix categorizing adversarial behavior by broad tactical goals, the UKC is more an 18-phase chronological model merging the CKC with *ATT&CK*. Therefore, it transforms a disconnected checklist of malicious *Linux* commands into an intrusion narrative. *MITRE ATT&CK* groups techniques under high-level tactics. The UKC has a finer granularity since it introduces specialized, transitional phases. It also highlights operational bottlenecks (e.g., attackers reusing `cron` for both persistence and privilege escalation), offering high-priority targets for detection.

This chapter applies the UKC framework to analyze the techniques observed across contemporary *Linux*-targeting intrusion campaigns. While trying to conceal it to the minimum, some repetition is present across multiple phases due to these attack phases often sharing the same technical mechanisms and system targets. Comprehensive definitions for all referenced malware can be found in the Glossary.

[1] https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html
[2] https://attack.mitre.org/
[3] https://www.fox-it.com/nl-en/
[4] https://attack.mitre.org/groups/G0007/

2.1. Reconnaissance

The initial reconnaissance phase consists of researching, identifying and selecting targets using active or passive reconnaissance [15].

2.1.1. Active Reconnaissance

Adversaries automate active reconnaissance using networking tools that are native to most *Linux* distributions (or often available in their repositories) [16] in order to build a map of the target's defenses and discover vulnerable services. Table 2.1 details the common active enumeration mechanisms.

Tool	**Operational Function**
`ping`	Verifies host connectivity via ICMP echo requests to determine target reachability and baseline firewall ACL filtering.
`traceroute`	Enumerates intermediate routing hops to isolate network topology and identify the edge/gateway firewall router.
`nslookup`	Performs DNS queries against target nameservers to resolve hostnames and infer the nature of running services.
`netcat` (`nc`)	Evaluates port state (open vs. closed/filtered) and probes individual endpoints to retrieve service banners (e.g., HTTP server headers via `GET /` or SMTP response banners via `help`).
`nmap -sn`	Executes a network-wide ping sweep to rapidly list active hosts responding to echo requests.
`nmap -sS`	Performs a TCP SYN stealth scan against the top 1,000 common ports to identify `open`/`closed`/`filtered` states and build firewall filtering rules.
`nmap -sV`	Identifies service names and exact version strings running on open ports to facilitate vulnerability matching.
`nmap -O`	Analyzes low-level TCP/IP stack implementation nuances to fingerprint the target OS.
`nmap -A`	Executes a comprehensive scan combining port enumeration, service version detection and OS fingerprinting in a single execution.

Table 2.1. Active Network Reconnaissance Techniques and Tooling

The persistence of these enumeration methodologies comes from the stability of core internet protocols. Because the underlying architecture remains governed by deterministic connection handshakes, static routing behaviors and standard-

ized packet headers, these legacy scanning techniques retain their effectiveness across modern enterprise networks.

2.1.2. Passive Reconnaissance

Passive reconnaissance techniques, as covered in [17], attempt to gain information about targeted computers and networks without actively engaging with the victim's infrastructure. Most of these methodologies are not *Linux*-specific. The most common set of techniques falls under the umbrella of Open source intelligence (OSINT), where attackers gather information from publicly available web sources. Collected information can include details about the target organization and its employees, such as email addresses, phone numbers, IP ranges, domain names, third-party suppliers, deployed technologies, geographic locations and social networking accounts. As TechTarget further notes, these activities are frequently automated using tools like *theHarvester*[5], to query public registries, and search engines like *Google* or *Bing*.

The same source explains that analyzing an organization's public website for administrative clues, such as exposed usernames or software version strings, represents a standard passive vector since public web servers are routinely accessed by the general public. Organizations often inadvertently expose metadata within public documents or page contents that reveal insights into their internal operating environments. For offline inspection, publicly accessible files can be retrieved in a non-disruptive manner, while external services such as *Netcraft* provide open-source metadata regarding website hosting providers, history and SSL/TLS certificates. Adversaries can also map network layouts and internet connectivity by reviewing public DNS infrastructure, including IP delegation, domain ownership and active zone records via the `whois` utility. Furthermore, specialized search engines like *Shodan* aggregate publicly exposed device banners, allowing attackers to identify internet-facing host vulnerabilities without interacting with the target network directly.

Finally, physical searches involve reviewing discarded assets or documentation, such as searching through unredacted trash or examining data remnants on discarded hard drives, as any improperly disposed item can present a potential corporate inroad [17].

[5] https://github.com/laramies/theharvester

2.2. Resource Development

Once the initial targeting information is gathered during the reconnaissance phase, attackers must prepare the infrastructure and tools required for the intrusion. Resource Development consists of these preparatory activities aimed at setting up the infrastructure required for the attack [15]. More than simply writing code, this phase requires building a complete operational foundation, from setting up external communication channels and deceptive lures to engineering payloads with built-in evasion mechanisms for the upcoming intrusion.

2.2.1. Malware Development and Adaptation

One common technique is to develop *Linux* variants of existing malware, such as FASTCash [18], SprySOCKS [19], RTM Locker [20] and Linodas [21]. For example, Linodas evolved from SimpleRemoter [21] and Gh0st RAT that primarily targeted *Windows* only. Another technique is to adapt existing *Linux* malware or malware parts, such as Auto-Color [22], [23], the Mélofée [24] malware family, Gomir [25] and Krasue [26]. Mélofée is based on an open-source *Linux* kernel rootktit named *Reptile* [24] and some malware, like Krasue [26], embeds multiple rootkits.

Threat actors frequently compile static operational data directly into malware binaries, embedding encrypted C2 configurations [22] and preparing architecture-specific builds to ensure compatibility across diverse target environments [27].

2.2.2. Infrastructure Setup and Staging

Attackers acquire and set up C2 servers and hosting infrastructure, which can involve registering new domain names ahead of time and setting up C2 configurations [28] or compromising legitimate websites to host payloads [29], [30].

Social engineering baits are meticulously prepared, such as spoofed government websites (e.g., mimicking India's Ministry of Defence press release portal for ClickFix campaigns [31]). These lures include decoy documents [30], [31], malicious executables [30], [32] and domain registration using typosquatting [31].

2.2.3. Evasion and Persistence Features

Malware binaries are often stripped of symbols during compilation to complicate static reverse engineering [20]. Threat actors also incorporate pre-execution anti-analysis logic, such as process masquerading routines, deliberate ELF header corruptions [1], *XOR*-based string encryption [33], [34] and environment verification or execution delays designed to evade sandboxes [34]. To ensure long-term access post-delivery, deployment scripts and payload installers are pre-configured with logic targeting system initialization points (e.g., `/etc/rc.local` or `/etc/init.d/`) [1], scheduled tasks (`cron`) [27], [35], symbolic linking routines [28], [36] or system triggers like `udev` rules [37]. Furthermore, payloads are frequently equipped with competitive routines, such as embedded termination lists and search scripts, engineered to neutralize rival malware occupying the same target environment [1], [33], [38], [39]. The runtime mechanics of these evasion and persistence mechanisms are covered in greater detail in Secs. 2.7 and 2.6.

2.2.4. Tooling

Attackers obtain and customize tools for various attack stages. This can involve using existing open-source projects as a base for their malware development [19], [21], [24], [26] or acquiring commercial tools like *Cobalt Strike* [24]. Aside from that, legitimate *Linux*-native tooling is identified to plan post-exploitation abuse. For instance, rather than building complete custom platforms, threat actors actively develop or acquire specialized userland rootkit modules designed to hook native diagnostic tools, such as `ls`, `ps`, `netstat`, `ldd`, `crontab`, and `top`, preparing the necessary capabilities to hide malicious activities upon deployment [21], [39].

2.3. Delivery

With infrastructure established and payloads adapted, adversaries must find a way to deploy these assets against the victim. In the UKC model, the Delivery phase consists of techniques resulting in the transmission of a weaponized object to the targeted environment [15]. In contemporary *Linux* threat landscapes, this transmission relies on a diverse spectrum of vectors, ranging from user-targeted social engineering and software supply chain compromises, to the direct targeting of exposed network services and the use of conditional, staged download mechanisms.

2.3.1. Phishing and Social Engineering

Phishing remains a common way to deliver malware. For example, the ClickFix campaign related to the *APT36* Pakistani group uses government imitating websites, to deliver cross-platform malware [31]. This involves directing users to fake *CAPTCHA* pages (e.g., `/captcha/windows.php`, `/captcha/linux.php`) and then silently copying a malicious command to the clipboard for the user to paste and execute [31]. The Interlock ransomware group has also adopted this ClickFix technique, delivering fake installers often disguised as browser or security software updates (e.g., `FortiClient.exe`, `Webex.exe`, `GlobalProtect.exe`) [29].

Some campaigns, like the one discussed by *Securonix* [32], are believed to have started with phishing emails that contained malicious files or links (e.g., `.zip`, `.rar`, `.iso`, `.pdf`), some being used to deploy custom-made emulated Quick Emulator (QEMU) *Linux* boxes.

SideCopy, a group similar to *APT36* [40], creates campaigns that spread via phishing links that download archive files, such as `Homosexuality - Indian Armed Forces.zip`, which contains malicious shortcut (`.lnk`) files for *Windows* and ELF executables for *Linux* [30].

2.3.2. Network Delivery to Exposed Services

Many ransomware families and botnets targeting *Linux* simplify their delivery phase by transmitting weaponized payloads directly to exposed, internet-facing services [35]. Instead of relying on user interaction like phishing, adversaries deliver malicious network packets or crafted requests targeting specific technologies, such as *IBM*

or *Oracle* databases [35]. For instance, the *8220 Gang* identifies potential entry points through broad internet scans and delivers initial payloads directly to the listening ports of vulnerable cloud applications [27]. Similarly, the *Betting on Bots* campaign delivered its malicious code via crafted *HTTP* requests sent to a server running *Apache2* [33]. In the case of the perfctl malware, the initial delivery mechanism targets message-queue services by transmitting crafted payloads aimed at the `CVE-2023-33246` vulnerability in *Apache RocketMQ* [39].

2.3.3. Supply Chain and Trojanized Software

North Korean espionage actors, such as the *Springtail* group (also named *Kimsuky*), favor software installation packages and updates, trojanized software installers and fake software installers as infection vectors [25].

Beyond installer spoofing, threat actors increasingly compromise open-source package registries to orchestrate developer-targeted supply chain attacks. For example, campaigns targeting the `npm` ecosystem use compromised developer credentials to inject malicious code into trusted packages via automated pre-installation scripts [41]. When a developer or automated CI/CD pipeline installs the dependency, the script triggers automatically in the background to profile the host environment, execute obfuscated payloads and establish covert persistence across *Linux* environments [41].

2.3.4. Staged Delivery Mechanisms

As a side technique, malicious actors frequently use native *Linux* command-line tools like `wget` and `curl` to download payloads onto compromised systems [27], [38], [39]. The *8220 Gang* sometimes uses *base64*-encoded Python `urllib` packages or custom bash functions for downloading, serving to obscure their activities and evade signature-based detection [27].

As explained in [39], the perfctl malware employs a clever technique during delivery. If the target user tries to download it without a specific, correct user agent, the download server will return an innocent integer, concealing the malware's presence. Following execution, it immediately transitions into process masquerading and binary replacement to ensure persistence, as detailed in Section 2.6.4.

2.4. Social Engineering

While some payloads are delivered via direct network exploitation, others require human interaction to bypass perimeter defenses. Social engineering consists of techniques aimed at the manipulation of people to perform unsafe actions [15]. To achieve this, threat actors deploy a range of psychological pretexts, from interactive browser-based deceptions to carefully crafted phishing operations using false personas and provocative baits.

2.4.1. ClickFix-style Social Engineering

As mentioned in Sec. 2.3, the ClickFix technique involves deceiving users into executing malicious commands by presenting fake system prompts or *CAPTCHA* verifications. Some security blogposts [29], [31] give more detail about how it works:

1. Attackers display spoofed *CAPTCHA* overlays, sometimes with deliberate misspellings (e.g., `"I'm not a rebot"`) to better evade automated scanning or impersonate legitimate services like *Cloudflare*.
2. Users are tricked into manually copying malicious shell commands to their clipboard and then pasting and executing them in a terminal. The attacker provides explicit, step-by-step instructions, such as `"Press ALT+F2"`, `"Press CTRL+V"` and `"Press ENTER"`. To enhance credibility and reduce suspicion, the background of these deceptive pages may feature faint watermarks from legitimate cybersecurity information sites.
3. To maintain the psychological ruse and prevent the user from realizing they have been compromised, the interaction frequently concludes with the display of a decoy document, such as a cloned government press release, reinforcing the illusion of authenticity.

2.4.2. Phishing and Spear Phishing

While the technical delivery of phishing payloads relies on email or malicious links, the social engineering component focuses on the pretexts used to establish trust and urgency. For instance, the `CRON#TRAP` campaign relied on convincing emails to trick users into manually extracting large, weaponized `.zip` archives [32]. Other

threat actors rely on highly targeted, psychological lures. The *SideCopy* group, for example, distributes malicious archive files using highly provocative, context-specific filenames (such as `Homosexuality - Indian Armed Forces.zip`) designed to exploit human curiosity and urgency [30]. Similarly, the *Springtail* espionage group executes highly targeted spear phishing campaigns by adopting credible false personas (i.e., journalists, academics, and industry experts) to manipulate specific high-value individuals into interacting with malicious assets [25].

2.5. Exploitation

Whether triggered by a deceived user or a direct network payload, the next critical step is to execute malicious code on the target. Exploitation consists of techniques to exploit vulnerabilities in systems that may, amongst others, result in code execution [15]. Within *Linux* environments, adversaries weaponize a broad range of system weaknesses to achieve this, including fundamental semantic flaws in the kernel architecture, exploits targeting exposed (web) services and internal privilege boundaries.

2.5.1. Semantic Vulnerabilities in Linux Kernel Security

Common semantic *Linux* kernel vulnerabilities can be separated into 10 classes [42].

Missing Pointer Checks: The kernel omits `access_ok` checks or misuses speculative operations such as `__get_user`, which fails to validate whether user-provided pointers or index variables reside strictly within user-space memory boundaries [42]. Unlike safer macros, `__get_user` directly retrieves data from a user-supplied pointer without checking its security perimeter, operating under the flawed assumption that memory validation has already been performed by preceding functions [43].

Missing Permission Checks: the kernel performs a privileged operation without checking whether the calling process has the privilege to do so. A vulnerability in this category results in a violation of a kernel security policy.

Buffer Overflow: the kernel incorrectly checks the upper or lower bound when accessing a buffer, allocates a smaller buffer than it is supposed to, uses unsafe string manipulation functions or defines local variables which are too large for the kernel stack.

Integer Overflow: the kernel performs an integer operation incorrectly, resulting in an integer overflow, underflow or sign error. The adversary can trick the kernel into using the incorrect value to allocate or access memory, allowing similar attacks as allowed by buffer overflow vulnerabilities.

Uninitialized Data: the kernel copies the contents of a kernel buffer to user space without zeroing unused fields, thus leaking potentially sensitive information to user processes, such as variables on the kernel stack.

Null Dereference: it is defined by OWASP [44] as a `NULL` pointer dereference that occurs when the application dereferences a pointer that it expects to be valid, but

is NULL, typically causing a crash or exit.

DIVIDE BY ZERO: self-explanatory.

INFINITE LOOP: self-explanatory.

DATA RACE/DEADLOCK: concurrent access of a same piece of data by multiple threads without proper coordination can lead to incorrect values being read or written or even cause the program to freeze if threads get stuck waiting on each other.

MEMORY MISMANAGEMENT: vulnerabilities in kernel memory management, such as extraneous memory consumption, memory leak, double free and use-after-free errors.

2.5.2. Exploiting Public-Facing Applications and Services

To breach initial perimeters, adversaries frequently target web infrastructure and enterprise applications through automated scanning and the weaponization of known vulnerabilities. Widely adopted software frameworks and web servers represent primary targets. For instance, the *8220 Gang* heavily exploits *Log4Shell* (CVE-2021-44228) to breach cloud infrastructure [27], while the Sysrv botnet and the *REF6138* campaign exploit specific *WordPress* instances and *Apache2* deployments, respectively [33], [45]. Apart from standard web servers, complex enterprise suites and administrative panels are aggressively targeted using both N-day and zero-day exploits. Indeed, *Earth Lusca* uses unauthenticated Remote Code Execution (RCE) flaws in *GitLab* (CVE-2021-22205) and *Zimbra* (CVE-2019-9670/ CVE-2019-9621) [19], while the IceFire ransomware targets exposed *IBM* services [35]. Furthermore, highly organized campaigns rapidly integrate novel exploits, demonstrated by *Ebury* operators using a zero-day pre-authentication flaw in Control Web Panel (CWP) (CVE-2021-45467) to compromise servers in bulk [36]. Other malware, such as perfctl, relies on opportunistic, broad-spectrum scanning to exploit a wide array of misconfigurations and vulnerable endpoints [39].

2.5.3. Local Privilege Escalation Exploits

Once initial access is achieved, attackers routinely deploy local privilege escalation exploits to obtain root privileges, primarily targeting user-space utilities and legacy kernel vulnerabilities. A favored user-space target is the *Polkit* component: both the *REF6138* campaign and the perfctl malware actively exploit the *PwnKit* vulnerability (CVE-2021-4034) to execute arbitrary code with root privileges [33], [39].

Alongside user-space flaws, kernel-level vulnerabilities remain heavily weaponized. Large-scale analyses of malicious ELF binaries, including tools used by the Ebury botnet, reveal that well-documented kernel flaws, most notably *Dirty COW* (`CVE-2016-5195`) and *Linux Overlay* FileSystem (FS) vulnerabilities (`CVE-2015-1328`), are frequently embedded directly into payloads to ensure reliable escalation across varying environments [1], [36].

2.6. Persistence

Once initial access is achieved through exploitation, attackers must ensure their foothold survives system reboots or configuration changes. Persistence consists of any access, action or change to a system that gives an attacker persistent presence on the system [15]. In *Linux* environments, adversaries achieve this durability by embedding their payloads deeply into the system architecture, exploiting everything from early boot sequences and automated job schedulers to dynamic library loading and device-event triggers.

2.6.1. Bootloader and Init System Manipulation

Within the context of the *MITRE ATT&CK* framework *Pre-OS Boot* technique (T1542), persistence can be achieved by hijacking the boot chain before user-space initialization completes. At the earliest stage, adversaries target the bootloader itself, tampering with Grand Unified Bootloader (GRUB) configuration files in `/etc/default/grub` or `/etc/grub.d/` to inject malicious kernel parameters (e.g., `init=/payload.sh`) or register rogue boot entries before invoking `update-grub` [46]. Moving further into the initialization sequence, attackers alter the temporary root FS (Initial RAM FileSystem (initramfs)) by unpacking, patching and rebuilding `initrd.img` or deploying utilities like `dracut`[6] to load malicious modules into early memory [46].

Once control transfers to the initialization subsystem, threat actors abuse runlevel scripts and service managers for persistent execution [1]. For example, *Earth Lusca*'s SprySOCKS [19] and Linodas [21] register persistent services via `systemd` or `chkconfig`, while *Iron Tiger*'s SysUpdate drops unit configurations directly into `/usr/lib/systemd/system/` [28]. Similarly, the perfctl malware establishes itself as a persistent background daemon, executing stealthily after unlinking its initial binary on disk [39].

2.6.2. Scheduled Tasks and Automation

Adversaries frequently abuse routine execution frameworks to establish persistent footholds, mapped in the *MITRE ATT&CK* matrix under the *Scheduled Task/Job* technique (T1053). On *Linux* hosts, scheduled job facilities represent a primary attack vector, allowing commands to run at predetermined intervals [1], [33], [45].

[6] https://wiki.archlinux.org/title/Dracut

Threat actors like the *8220 Gang* deploy redundant `cron` tasks to maintain access [27], while variants such as KAIJI directly tamper with system-wide schedules in `/etc/crontab` [33]. Notably, low-privileged actors can still schedule persistence without root privileges by exploiting the `setuid` permissions of the standard `crontab` binary [1].

Complementing periodic schedulers, attackers hook into interactive user environments and authentication mechanisms to trigger or permit execution upon login. This is achieved by injecting commands inside shell initialization scripts such as `~/.bashrc`, `~/.bash_profile` or `~/.profile`, a technique used by perfctl to launch itself during session initialization [39]. On desktop-oriented *Linux* environments, this persistence extends to graphical sessions, where malware injects `autostart` entries into `.desktop` application files [1]. Furthermore, to ensure direct administrative access, if a pre-existing backdoor account is not found, some malware directly writes new backdoor user accounts into `/etc/passwd` and `/etc/shadow` [1]. Similarly, malware such as Tsunami establishes silent backdoor access by appending a new public key directly to the user's `~/.ssh/authorized_keys` file, enabling persistent passwordless authentication for the attacker [47]. Finally, the `GSOCKET` tool facilitates persistence through shell profile modification, run control (`rc.local`) and `systemd` configurations [33].

2.6.3. System Process Modification

Under the *MITRE ATT&CK Create or Modify System Process* technique (`T1543`), adversaries hook system services to maintain access. Adversaries can create custom *D-Bus* service files to establish permissive configurations that initiate reverse connections with root permissions [46]. Closely aligned with these service hooks is the *MITRE ATT&CK Event Triggered Execution* technique (`T1546`), in which attackers place or modify custom scripts inside `/etc/NetworkManager/dispatcher.d/` [46]. These dispatcher scripts run payloads with root privileges whenever specific network events trigger, such as a network interface transitioning up or down [46].

2.6.4. File System Manipulation and Infection

Persistence is also achieved through binary tampering, dynamic linking manipulation and hardware-event hooks. Malware can replace legitimate system applications or infect existing ELF binaries on the host [1]. Shared library hijacking represents a stealthier alternative. For example, the Ebury backdoor hijacks legitimate libraries such as `libkeyutils.so.1` by injecting a `DT_NEEDED` entry into the

dynamic section, forcing the malicious library to load without altering the original file size [36]. Similarly, the related Ebury's *FrizzySteal* module injects itself into `libcurl` by modifying `libz.so.1` [36].

Rootkit-based persistence operates across both user and kernel space using distinct mechanisms. The Auto-Color malware modifies `/etc/ld.preload` to force its library (`libcext.so.2`) into every running process [23]. For kernel-level execution, Mélofée's rootkit installer adds persistence entries for its kernel module to `rc.modules` under `/etc` [24], while Krasue embeds multiple precompiled rootkit variants that load into kernel space via `init_module` [26]. Finally, malware can exploit device-event triggers. Indeed, the sedexp malware uses custom `udev` rules to execute its payload whenever a targeted device event occurs, such as the addition of `/dev/random` [37].

2.7. Defense Evasion

To protect their newly established persistent access from security mechanisms and administrators, threat actors actively hide their tracks. Defense Evasion consists of techniques an attacker may specifically use for evading detection or avoiding other defenses [15]. On *Linux* systems, adversaries achieve this stealth through a multi-layered approach, employing everything from binary obfuscation and anti-debugging mechanisms to process masquerading, network traffic manipulation, and advanced kernel-level rootkits.

2.7.1. Code and Binary Manipulation

Attackers modify code and binary structures through obfuscation, encryption and compiler-level tampering to complicate analysis and detection. Despite these capabilities, large-scale studies note that the overall obfuscation complexity of *Linux* malware generally remains lower than its *Windows* counterparts, with many developers neglecting basic steps like stripping symbols [1]. At the binary and execution layer, adversaries employ software packing to prevent static analysis. Because relatively few ELF packers exist beyond Ultimate Packer for Executables (UPX), developers frequently deploy modified UPX variants with altered magic numbers, internal strings or junk bytes designed to break automated unpackers [1], [33]. In rare instances, custom packing mechanisms are used, such as spawning a separate *Perl* interpreter process to dynamically decrypt and execute instructions [1]. Code and metadata are further concealed by encrypting sensitive data or payloads, obscuring executed commands [33] and dynamically resolving external Application Programming Interface (API) addresses at runtime [36]. To evade hash-based detection, attackers use binary padding, replacing the field named `NT_GNU_BUILD_ID` in ELF headers with randomized values [36].

Obfuscation extends to program logic, strings and symbols. Sensitive strings and data blocks are encrypted using various *XOR* schemes to minimize their visibility [34], while entire function bodies can be encrypted with algorithms like *RC4* to defeat static and memory analysis [48]. Exported function names are often replaced with alternative naming schemes to mask their true purpose [34] or mixed with deceptive names to obscure helpful context for reverse engineers. Beyond name scrambling, binaries are stripped of their symbols and symbol tables (`.symtab`), either via compiler options like `-s` or post-compilation utilities, to eliminate function identifiers [48].

Attackers also use build-time optimizations and file format anomalies to disrupt disassemblers. Aggressive optimization flags (e.g., `-O3 -funroll-loops`) expand function size and complexity by increasing the number of branches in the code, raising the reverse engineering effort [48]. Adversaries also add fake section headers, which causes disassemblers such as *IDA* to fail during function analysis and produce flawed disassembly. Similarly, overlapping the program entry point with the `.init` section leads analysis tools to misinterpret instructions [48].

2.7.2. Obstructing Code Flow and Analysis

These techniques aim to mislead disassemblers and analysts about the true execution path. Code reorganization alters the compilation method to automate obfuscation, while sensitive data is protected using stack strings or *XOR*-encoded stack strings that remain unreadable without the corresponding key [48].

Control flow is further obscured by severing direct relationships between routines during static analysis. Adversaries replace direct calls with indirect function calls (such as function pointers) or rely on signals (e.g., `SIGUSR1`) to trigger execution, which forces reverse engineers to trace `sigaction()` calls to reconstruct the control flow [48]. Similarly, early returns use assembly sequences like `push %0;ret` to transfer execution to an unexpected location mid-function, often coupled with dynamically calculated return addresses, to hide code segments from static inspection [48].

Finally, binary code stream manipulation directly breaks disassembler decoding logic since injecting non-code bytes (such as `.string` directives) may halt linear disassemblers and stops further analysis, while conditional jumps over an invalid byte prevent tools from correctly tracing the flow of execution [48]. Attackers also try to mislead the tools that analyze their code by inserting uncommon patterns (like two consecutive `jmp` instructions) to force these tools down the wrong code paths [48]. Additionally, they overlap instructions in memory, tricking disassemblers into misreading where instructions start and end, which completely hides the real malicious code [48].

2.7.3. Evasion and Anti-Debugging

Threat actors deploy specialized techniques to detect and obstruct debugging, reverse engineering and sandbox analysis environments. To counter interactive debugging, binaries perform checks early in execution before `main()` is invoked.

These routines misuse `ptrace` (such as issuing `PTRACE_TRACEME` or `PTRACE_ATTACH` on themselves) to exploit the constraint that only one process can attach at a time, inspect `/proc/self/status` to identify tracing debuggers [48] and explicitly avoid injecting into programs running within a debugger like GDB [36]. To detect tampering or patched debugger checks, malware calculates checksums over critical code sections [48]. Post-mortem memory analysis is similarly obstructed by calling `prctl` to disable core dumping entirely or by using `madvise()` with `MADV_DONTDUMP` to exclude specific memory ranges, such as deobfuscated code, from core dumps [48].

For stealth on compromised systems, kernel-mode and userland rootkits conceal files, directories, processes, network connections and general malware activity by hooking system calls (e.g., `kill()`, network functions and file listing) and altering basic system utility outputs [1], [24], [26], [39].

In order to block automated dynamic analysis, malware tests runtime environments using verification mechanisms like checksums and `/boot`-targeted anti-sandbox technologies [34]. Binaries actively probe for virtualization artifacts across system paths (including `/sys/class/dmi/id/`, `/proc/cpuinfo`, `/proc/sysinfo`, `/proc/scsi/scsi`, `/proc/vz`, `/proc/bc`, `/proc/xen/capabilities` and `/proc/<PID>/mountinfo`) [1] and inspect memory size to detect resource-constrained sandboxes [29]. Malware also employs conditional anti-execution, such as verifying that its current filename matches a hardcoded string to evade sandboxes that rename samples [1]. Finally, execution is delayed by running useless time-delaying code [1] or flooding the system with thousands of harmless APIs calls [34], since automated security sandboxes only analyze files for a few brief minutes before flagging software as safe if no malicious behaviour was encountered.

2.7.4. System Configuration and Process Hiding

Adversaries modify system settings and employ masquerading techniques to avoid detection and hinder forensic analysis. To weaken host defenses, attackers disable security features such as *SELinux* via `setenforce 0`, perform User Account Control (UAC) bypasses and disable or uninstall cloud security services and monitoring agents (e.g., *Alibaba* and *Aliyun*) [27]. Firewall controls are similarly altered or bypassed by disabling the firewall entirely (e.g., `ufw disable`), setting `IPTABLES` to allow all traffic or injecting rules to permit malicious incoming connections. Furthermore, malware tampers with critical system files, such as stripping immutable and append-only attributes from `/etc/ld.so.preload` and clearing its contents [27], while terminating competing cryptocurrency miners and rival malware with tools like `pkill` to retain resource control [1].

At the process level, binaries disguise their names to mimic standard system processes, such as `kdevtmpfs`, `ntfsys`, `pmtad`, `sshd`, `telnetd`, `cron`, `sh`, `busybox` or randomized names [1]. On the FS, malware employs benign-sounding file names like Auto-Color [22] and hides malware-related files and directories from standard directory listings [1]. Furthermore, perfctl copies itself from memory into deceptive disk locations (e.g., `/root/.config/cron/perfcc`, `/usr/bin/perfcc`, `/usr/lib/libpprocps.so` and `/usr/lib/libfsnkdev.so`), masquerading as legitimate system files and deleting its initial binary to obfuscate execution flow [39]. To further conceal their presence, binaries can wrap legitimate system utilities (such as `who`, `netstat` and `ps`) with malicious filtering modules (e.g., `.so6`) to sanitize their output and hide malware artifacts [21]. Similarly, to frustrate security analysts performing local host inspection, attackers deploy trojanized versions of standard system utilities, such as `crontab`, `lsof` and `top`, to actively conceal artifacts of malicious activity [39]. Finally, evidence of compromise is eliminated through timestomping to alter file timestamps, disabling shell history logging to prevent command recording [24] and clearing *Linux* system logs, audit logs and bash histories [36].

2.7.5. Network Evasion

Adversaries employ diverse techniques to conceal network activities, exfiltrate data and obscure external communications. At the transport layer, malware hides C2 connections and configuration metadata using proprietary encryption algorithms or advanced methods inspired by the Symbiote malware [22], [49]. Communication channels are further secured by combining asymmetric (*RSA*) and symmetric (*RC*) encryption schemes [34]. To disguise exfiltration endpoints, dynamic DNS resolution techniques calculate and decrypt the target exfiltration server's IP address from `DNS TXT` records rather than relying on standard `A` records [36]. On the local host, attackers manipulate *Netfilter* rules via a modified `iptables` binary, effectively hiding malicious network traffic from system administrators [36].

At the application layer, in order to generate illicit revenue from stolen credentials and redirected web traffic, threat actors such as the *Ebury* group have transitioned from modifying core web server files to installing custom *Apache* and *Nginx* modules to proxy raw traffic, intercept requests and redirect visitors [36]. Specific implementations include Ebury's *HelimodSteal* module, which hooks web server processing by adding an input filter to capture and exfiltrate sensitive data submitted via `HTTP POST` forms [36]. For redirection, Ebury's *HelimodRedirect* module registers an output filter to divert visitors to external sites, often redirecting only a small fraction of traffic to evade detection, while Ebury's *KernelRedirect* module intercepts traffic at the *Linux* kernel level, hooking the network subsystem to con-

ditionally alter outbound `Location` headers and transparently reroute a randomized subset of users [36].

2.7.6. Kernel-Level Evasion (eBPF)

APT groups have adopted modern evasion methods by abusing BPF filters to facilitate backdoor deployment [4], [9]. By using the eBPF technology to manipulate system visibility, this type of malware can conceal running processes while obfuscating its network communications and harvesting sensitive environment metadata [9]. This level of kernel-level tampering effectively neutralizes host-based monitoring tools and blinds defenders. As detailed in 1.4.1, this initial eBPF concealment might serve as the first phase of a multi-stage compromise, enabling the installation of a traditional LKM rootkit that provides deeper system evasion by hiding specific malicious files and directories [9].

2.8. Command & Control

With a stealthy and persistent presence secured, the malware must establish a reliable communication channel back to its operators. Command & Control consists of techniques that allow attackers to communicate with controlled systems within a target network [15]. In *Linux* environments, adversaries operationalize this phase by establishing resilient network protocols, obfuscating data in transit, and deploying frameworks for interactive remote control and internal process coordination.

2.8.1. Establishing Communication Channels

Adversaries build resilient C2 channels by combining multi-layer protocols with advanced evasion techniques. At the network and transport layers, attackers use a variety of communication channels, including raw TCP sockets, TLS-encrypted tunnels and UDP sockets, the latter frequently running the Keep Connection Protocol (KCP) protocol [24]. To evade detection, this traffic is often routed over non-standard ports [24], [33]. Furthermore, threat actors disguise their C2 channels as kernel processes, including the use of encrypted connections disguised as kernel processes [33].

At the application layer, malware usea standard web protocols to facilitate C2 interaction, such as HTTP-based exchanges [24] and transmitting gathered system information to remote servers via `HTTP POST` requests [29], [33]. Adversaries bypass access restrictions on isolated targets by tunneling traffic through a controlled intermediary. For example, the *Chisel* utility is deployed to pass data covertly through firewalls using *WebSockets* [32].

2.8.2. Obfuscation and Evasion in Communication

Adversaries obfuscate the command, control and exfiltration flows through multi-layered encryption, data encoding, covert channeling and local log tampering. For transmission security, malware frequently employs proprietary encryption algorithms to obscure communication and configuration data [22], ranging from *RC4* with hardcoded keys [24] to custom *XOR* operations [34]. These communication channels may use unique, one-time keys generated dynamically for each individual message [22]. Payloads and operational commands are routinely *base64*-encoded to obscure ongoing activity and evade signature-based detection mech-

anisms [24], [27], [38], [50].

To bypass network inspection, some implementations construct covert transmission channels. These include injecting data directly into the initial `TCP SYN` packet or use the BPF engine for stealthy communication [34]. Alternatively, C2 exchanges can route through `DNS TXT` requests by encoding payload data inside DNS queries [36]. Furthermore, *Iron Tiger*'s SysUpdate malware implements DNS tunneling for its C2 communications, providing a covert channel that masks the true destination of the traffic [28].

2.8.3. Command Execution and Capabilities

Once communication is established, C2 frameworks provide adversaries with the capability to maintain interactive control and receive tasking. To secure this access across the infrastructure, malware families such as Auto-Color, Linodas, ShellBot, sedexp and Bvp47 allocate pipe-based communication channels to spawn internal remote shells [21], [22], [34], [37], [51]. These tools allow operators to interact directly with the compromised host. To maintain this connection stealthily, malware typically relies on beaconing behavior, gathering basic host telemetry (including OS versions, hostnames, IP addresses and active process lists) to transmit back to the C2 server [19]. These periodic "alive pings" can be disguised using atypical protocol messages, such as Real Time Streaming Protocol (RTSP) `DESCRIBE` requests [26].

Beyond interactive shells and beaconing, the C2 infrastructure serves as the primary conduit for lifecycle management and network routing. For operational maintenance, the server uses the established channel to push software updates, distribute new C2 URLs for resilience [21], [47] or issue self-uninstall and termination routines to erase the malware's footprint [22]. Finally, controlled machines can be repurposed as C2 relays, functioning as middleman proxies that route connections between remote targets and arbitrary argument-supplied IP addresses to further obfuscate the attacker's origin [22].

2.8.4. Internal Communication

Internal Inter-Process Communication (IPC) and remote tasking coordinate execution and maintain an ongoing access. Within the malware itself, internal communication can be handled via *UNIX* sockets, writing to and reading from files located in temporary directories to manage operational data [39]. Alternatively, cer-

tain *Linux* malware communicate with their rootkit components through signals (such as intercepting the `kill()` system call) instead of conventional IPC mechanisms, allowing the delivery of commands to hide processes, conceal ports or grant root privileges [26].

2.9. Pivoting

After establishing a secure communication channel, attackers often use the compromised host as a relay point to reach deeper into the network. Pivoting consists of tunneling traffic through a controlled system to other systems that are not directly accessible [15]. In contemporary *Linux* intrusions, the mechanics of Pivoting are linked with Lateral Movement. Because advanced implants use the exact same tunneling protocols and SSH Adversary-in-the-Middle (AitM) attacks to achieve both, the specific techniques for network traversal and proxy establishment are consolidated in Sec. 2.14.

2.10. Discovery

Before moving further or executing final objectives, the adversary must map the internal environment to identify valuable assets and network topologies. Discovery consists of techniques that allow an attacker to gain knowledge about a system and its network environment [15]. Within *Linux* ecosystems, adversaries accomplish this by parsing local files, enumerating running processes, querying network states, and monitoring user sessions to build a comprehensive profile of the compromised host.

2.10.1. System and Host Information Gathering

Adversaries extensively query compromised systems to profile host configurations, underlying hardware and current system activity [15]. Initial discovery focuses on basic system information, collecting details such as hostnames, usernames, CPU and RAM specifications, network interfaces, IPv4 and IPv6 addresses, kernel versions and machine architecture [15], [39]. For example, to ensure compatibility before deploying secondary tools, threat actors such as the *8220 Gang* execute native shell commands like `uname -m` to determine the specific machine architecture and deliver matching Tsunami payloads [27]. To extract detailed hardware, networking and kernel data, malware inspect virtual FSs, particularly `/proc` and `sysfs` [1], [21], by reading nodes such as `/proc/version`, `/proc/cpuinfo`, `/proc/meminfo`, `/proc/stat` and `/proc/net/tcp` [1].

Reconnaissance further extends across local system configurations, user records and FS state. Attackers parse essential files in `/etc`, including `/etc/passwd` to enumerate registered users [1], [38], as well as `/etc/resolv.conf` and `/etc/hosts` for DNS resolver settings and host-to-IP mappings [1]. Malware such as perfctl specifically hunt for exposed credentials and configuration files [39]. Historical and active session data is harvested from user activity logs, including `/var/run/utmp`, `/var/log/wtmp` and `/var/log/lastlog` [21]. Finally, adversaries execute native shell commands like `uname -a`, `df` and `ls -hal` to verify environment parameters and confirm file locations [32].

2.10.2. Process and Service Enumeration

To determine the runtime state of a compromised system, adversaries enumerate active processes, background services and user activity [21]. To achieve that,

they use standard built-in utilities like the `ps aux` command frequently invoked to capture granular process metadata, including process IDs and execution privileges [1], [38], while `who` is executed to monitor logged-in users [21].

Beyond standard native commands, threat actors introduce specialized tooling and custom scripts to deepen inspection and bypass privilege constraints. Utilities such as `pspy64`[7] are deployed to monitor executing processes and `cron` job tasks stealthily without requiring root privileges [33]. Additionally, adversaries list the current user's `crontab` entries and deploy custom scripts (e.g., `whatserver.sh`[8]) to aggregate server metrics, including active listening services, CPU load and memory utilization [33].

2.10.3. Network Reconnaissance

Network discovery can be conducted actively through direct interaction with the target or passively by gathering publicly available information [16], [17].

In active operations, adversaries map reachability, routing and local network state using native active reconnaissance tools listed in SubSec. 2.1.1. Aside from that, the `arp -a` command is used to collect the local system's ARP table [29]. For broader network reconnaissance, attackers deploy scanners such as `masscan` and `spirit` to scan IP ranges for specific open ports, including SSH [27]. Furthermore, threat actors use directory traversal fuzzing to seek out mistakenly exposed configuration files and secrets [39]. Upon successful exploitation, they establish connections to IP addresses tied to malicious frameworks, such as *Perfctl*, or *IRCbots* associated with the 8220 Gang [27].

On the other side, passive reconnaissance avoids direct contact by relying on OSINT and external asset indexing [17], as already stated in SubSec. 2.1.2. Attackers can also evaluate external network infrastructure by eavesdropping on accessible network traffic with *Wireshark* [17].

2.10.4. Credential and File System Discovery

Attackers seek out and access credentials and sensitive files to facilitate lateral movement and persistence [45]. Initial harvesting focuses heavily on remote access artifacts: adversaries enumerate SSH private keys, inspect `known_hosts` files

[7] https://github.com/dominicbreuker/pspy

[8] https://github.com/hackerschoice/thc-tips-tricks-hacks-cheat-sheet/blob/master/tools/whatserver.sh

to identify associated usernames [45] and parse command history files such as `~/.bash_history` to uncover previously established SSH connections [27]. Beyond local file parsing, threat actors extract credentials by directly dumping SSH private keys or harvesting passwords left on disk by other credential-stealing tools and then using *xCAT* commands to exfiltrate these stolen credentials over active SSH sessions [36].

2.10.5. Anti-Analysis-Related Discovery

While standard anti-sandbox discovery is covered under Sec. 2.7.3, the `perfctl` malware shows how modern threats execute discovery techniques against live administrators. To evade live inspection, this malware implements session monitoring by tracking login entries in `utmp` or `btmp` files. Upon detecting a newly logged-in user, it suspends its noisy operations, only resuming execution once the system returns to an idle state [39].

2.11. Privilege Escalation

While initial access provides a foothold, adversaries frequently land in restricted user contexts and must exploit local vulnerabilities or system misconfigurations to gain unrestricted control. Privilege Escalation consists of the result of techniques that provide an attacker with higher permissions on a system or network [15]. In *Linux* architectures, attackers secure these elevated permissions by exploiting known kernel and user-space vulnerabilities, abusing misconfigured system policies, and manipulating process credentials at runtime.

2.11.1. Exploiting Known Vulnerabilities

Attackers frequently target established flaws in the *Linux* kernel and critical system components to gain elevated access [1], [33]. Initial compromise often originates from exploiting unpatched vulnerabilities in public-facing applications or exposed services, such as web administration tools, which serves as a precursor to further privilege escalation [36]. For actual privilege elevation, threat actors use critical flaws like the *Polkit* vulnerability found in the *pkexec* component of the *PolicyKit* package, which has been exploited in the *Betting on Bots* campaign and by the sedexp malware to execute arbitrary code with root privileges [33], [39], [46]. Additionally, the *Ebury* gang operators rely on the *Dirty COW* vulnerability (`CVE-2016-5195`), cited as one of the most frequently exploited kernel vulnerabilities for privilege escalation [1], [36]. They weaponize this exploit to deploy a custom payload that uses the `chmod` system call to add the *setuid* bit to legitimate executables like *Perl*. By doing so, they abuse the native binary to execute arbitrary commands as the root user [1].

2.11.2. Abusing System Mechanisms

Attackers exploit intended system functionalities and configuration settings to achieve persistent, privileged execution [46]. By manipulating inter-process communication and authorization policies, adversaries deploy malicious *Polkit* rules, packaged as `.pkla` (old format) or `.rules` (new format) files, to bypass standard checks, enabling unprivileged users to execute privileged actions or obtain root permissions directly [46]. Similarly, threat actors create or abuse overly permissive *D-Bus* policies and vulnerable services to invoke privileged methods, execute arbitrary code and potentially establish reverse connections with root privileges [46]. This broader strategy of repurposing misconfigured native system bi-

naries and features for privilege escalation is cataloged by the *GTFOBins*[9] project, which serves as a primary resource for tracking these vectors in *Linux* environments.

Persistence and execution control are also secured through boot and network event mechanisms. Attackers manipulate bootloaders by injecting malicious kernel parameters (e.g., `init=/payload.sh`) into GRUB configuration files, altering existing boot menu entries or modifying the initramfs to trigger root-level code execution early in the boot sequence [46]. Furthermore, adversaries utilize *NetworkManager* dispatcher scripts by placing malicious files within `/etc/NetworkManager/dispatcher.d/`, ensuring payloads run automatically with root permissions in response to specific network events [46].

2.11.3. Credential and Process Manipulation

Direct manipulation of user and process identifiers facilitates privilege escalation [26], [39]. To circumvent local access controls, malware like perfctl hook *Libpam* symbols (e.g., `pam_authenticate`) to entirely bypass password verification during the authentication workflow. This allows unprivileged processes to execute unauthorized actions with elevated permissions [39].

Adversaries also escalate and maintain elevated control through kernel-level process manipulation and rootkit communication. Once an entry point is established, threat actors deploy specialized kernel-mode components that interact directly with process memory. For instance, Krasue's kernel rootkit not only conceals its presence but also monitors the system for specific commands or signals. After receiving them, it dynamically alters the credentials of the requesting unprivileged process, granting it root privileges [26].

2.11.4. Targeting Virtualization Platforms

Ransomware targeting virtualization environments require elevated privileges to achieve their destructive objectives, though the ransomware payloads themselves often rely on attackers to secure root access prior to deployment [20], [52]. Once executed with the necessary permissions, ransomware variants like RTM Locker and the *Linux* variant of Royal ransomware interact directly with the hypervisor management tools. They execute specific commands, such as `esxcli vm process list` and `esxcli vm process kill`, to forcefully enumerate and terminate running

[9] https://gtfobins.org/

virtual machines on Elastic Sky X integrated (ESXi) hosts [20], [35], [52]. This ensures that the underlying virtual disk files (e.g., `.vmdk`, `.vmem`, `.vswp`) are unlocked and available for encryption.

2.12. Execution

Using elevated privileges, adversaries can run arbitrary commands or deploy secondary payloads across the environment. The Execution phase consists of techniques that result in execution of attacker-controlled code on a local or remote system [15]. On compromised *Linux* hosts, adversaries operationalize this phase by abusing built-in scripting interpreters, deploying and launching compiled binaries, and injecting malicious payloads directly into live process memory.

2.12.1. Command and Scripting Interpreter Abuse

Attackers routinely abuse built-in *Linux* shell utilities and scripting interpreters, such as *Bash* and *Perl*, to execute code and orchestrate the next attack stages [1], [36], [38]. To stage and deliver initial payloads, adversaries employ diverse shell scripts built upon utilities like `wget`, `curl`, `lwp-download`, *Python*'s `urllib` package and custom *base64*-encoded *Bash* functions to retrieve and execute components such as coinminers and the Tsunami *IRCBot* [27]. Once a host is compromised, malware such as Auto-Color and *Springtail*'s Gomir backdoor facilitates ongoing control by executing arbitrary string commands or spawning full interactive shells [22], [23], [25].

Some execution strategies also use advanced scripting environments and user-assisted techniques. In the CRON#TRAP campaign, threat actors use *PowerShell* to launch a custom lightweight *Linux* environment emulated via QEMU, which then executes a *Go*-based *Chisel* client[10] to establish a persistent websocket backdoor [32]. In contrast, the *APT36*-linked ClickFix campaign relies on social engineering to persuade users into manually pasting and running malicious command sequences from their clipboard that invoke `curl`, `chmod` and `bash` on *Linux* (or execute via `mshta.exe` on *Windows*) [31].

2.12.2. File-Based Execution

As a primary deployment tactic, attackers manipulate file structures and use native utilities to guarantee payload execution [36], [47]. Once initial access is achieved or a dropper is triggered, threat actors routinely download malicious compiled binaries to the disk, grant them execution privileges via the `chmod` utility and immediately launch them to initiate the core infection routine [33], [39], [47].

[10] https://github.com/jpillora/chisel

2.12.3. Process Manipulation and Injection

Rather than executing standalone binaries, sophisticated adversaries manipulate existing system processes to execute malicious code directly within the memory space of trusted applications [1], [36]. Threat actors achieve this by using system calls such as `ptrace` or `dlopen` hooks to inject shared libraries or arbitrary payloads into running processes, hijacking the host application's execution flow to run the attacker's instructions [1]. Advanced malware families manipulate process execution at the thread level to optimize the speed and efficiency of their payloads. For instance, ransomware families like RTM Locker deploy custom functions (such as `pthread_wrapper_main`) to spawn multiple concurrent threads for their encryption routines [20].

2.13. Credential Access

Localized executions frequently aim to harvest authentication material, which is critical for bypassing upcoming security boundaries. Credential Access consists of techniques resulting in the access of, or control over, system, service or domain credentials [15]. In *Linux* environments, attackers systematically harvest these secrets by intercepting SSH authentication flows, dumping system credential files and hooking native memory libraries to capture data in transit.

2.13.1. SSH Credential Theft and Manipulation

SSH credentials remain a primary target for adversaries seeking lateral movement and administrative control [36]. To harvest valid account combinations, threat actors routinely scan for open port 22 and execute dictionary or brute-force attacks against poorly secured SSH servers [51]. In more sophisticated operations, the *Ebury* group conducts SSH AitM attacks by launching ARP cache poisoning from compromised hosts to intercept SSH traffic from valuable targets [36]. These operations directly harvest credentials, dump SSH private keys and exfiltrate the *Bash* command history. Attackers also routinely dump SSH private keys from compromised machines to use them later [36].

Credential interception is further achieved through deep client and server manipulation. The Ebury malware hooks functions within the *OpenSSH* client (such as using `DT_NEEDED` entries in `libkeyutils.so`) or injects itself into interactive shells to capture user-entered credentials, relaying the intercepted data to the main Ebury process over a *UNIX* socket. Operators can then connect to an Ebury-compromised *OpenSSH* server and invoke a custom *xCAT* command to retrieve the stored credential pool [36]. To maximize their credential harvest, threat actors also extract and attempt to crack hashed hostnames from *OpenSSH* `known_hosts` files, building a repository of valid network targets and authentication material for future attacks [36].

2.13.2. System File Manipulation and Authentication Bypass

Attackers systematically target critical system files that store credential data and authentication logic to harvest accounts and sustain access [1], [39]. To extract

credentials, malware accesses and dumps the contents of `/etc/passwd` and `/etc/shadow` to obtain user lists and hashed passwords. For example, the *Ebury* group specifically dumps the `/etc/shadow` file and applies brute-forcing techniques to recover passwords [1], [36]. Beyond static file tampering, adversaries compromise runtime authentication to intercept passwords in transit. The perfctl malware, for instance, hooks *Libpam* functions, specifically `pam_authenticate`, to secretly log and harvest user credentials as they are entered into the system [39].

2.13.3. Deployment of Stealers & Network Sniffing

Specialized malware and network techniques are deployed to capture credentials both in transit and directly from the disk. Threat actors use dedicated infostealing malware: ransomware groups like *Interlock* deploy stealers such as *LummaStealer* and *BerserkStealer* to harvest sensitive data [29], while groups like *Earth Lusca* specifically aim to exfiltrate email account credentials as part of their espionage campaigns [19]. On compromised hosts, the *Ebury* group takes this further by reverse-engineering and decrypting credentials left on disk by other SSH password-stealing tools, hijacking the credential-harvesting efforts of rival threat actors [36].

Adversaries also intercept live network traffic to capture secrets before they are protected by network encryption. Ebury-associated implants (such as *HelimodSteal* and *FrizzySteal*) inject into applications relying on the `libcurl` library to intercept sensitive `HTTP POST` payloads [36]. This deep hooking capability allows attackers to harvest credentials, session tokens and credit card details directly from the memory of compromised web servers, successfully capturing the data even when the original traffic is secured via HTTPS. Furthermore, to expand their reach across hosting infrastructure, attackers exploit software vulnerabilities to harvest plaintext passwords directly from administrative provisioning systems [36].

2.14. Lateral Movement

After securing credentials on a single host, threat actors reuse them to expand their control across the broader network infrastructure. Lateral Movement consists of techniques that enable an adversary to horizontally access and control other remote systems [15]. In *Linux* environments, attackers execute this horizontal expansion by reusing intercepted SSH credentials, establishing internal network proxies, and hijacking live administrative sessions to pivot seamlessly across the infrastructure.

2.14.1. Credential Reuse and Stolen Credentials

Once an initial foothold is established and local credentials have been harvested, adversaries use this stolen authentication material to pivot to adjacent systems. Rather than relying heavily on new exploits, threat actors frequently reuse intercepted plaintext passwords, compromised SSH private keys and trusted network mappings extracted from `known_hosts` files to authenticate against internal servers seamlessly [36]. By using these legitimate administrative credentials, attackers can pivot to adjacent servers via standard SSH connections, expanding their reach across the infrastructure while remaining largely undetected by security mechanisms.

2.14.2. Abusing Remote Services

To execute commands and establish interactive control over adjacent systems, threat actors heavily abuse legitimate internal remote services. SSH remains the primary vehicle for this lateral expansion. Attackers exploit existing trust relationships between internal servers by using compromised `authorized_keys` configurations, allowing them to authenticate to new internal targets without requiring passwords [47]. Once a connection is established, adversaries can deploy backdoor tools like ShellBot to gain remote shell access and conduct internal port scanning across the network [51].

2.14.3. Pivoting and Tunneling

Compromised hosts are used as relays to reach internal targets across the victim infrastructure [22], [34]. Threat actors configure these infected endpoints as jump servers or proxies using malware such as Bvp47, Auto-Color and Mélofée to launch attacks against other internal systems [22], [24], [34]. In particular, Auto-Color and Mélofée implement built-in network proxy capabilities that transform the compromised host into a middleman for reaching internal resources [22], [24]. Beyond proxying, adversaries use tunneling to bridge internal network segments. This technique is active across operations like the CRON#TRAP campaign and attacks involving the KAIJI malware [32], [33].

2.14.4. Execution of Commands and Payload Deployment

Attackers execute commands and deploy tools across remote systems to expand control throughout the target environment [50]. File transfer mechanisms support this process: internal file transfers enable the delivery of secondary tools and payload distribution across the network [21], [50], [51].

To navigate heterogeneous environments, threat actors employ cross-platform malware and dedicated post-exploitation frameworks. For example, SideCopy's deployment of *Ares* RAT across both *Windows* and *Linux* allows adversaries to compromise diverse OSs within the network simultaneously [30], [34]. Furthermore, some attackers use post-exploitation frameworks like *Cobalt Strike*, capitalizing on their established capabilities for lateral movement [19], [24].

2.14.5. AitM Attacks

The *Ebury* operators employ internal SSH AitM attacks to pivot to specific high-value targets. With a compromised host within the target's network segment, adversaries use techniques such as ARP spoofing combined with `iptables` rules to redirect internal network traffic through an attacker-controlled system. By intercepting these active administrative connections, the attackers hijack the session to facilitate the installation of Ebury onto adjacent targeted servers [36].

2.15. Collection

Once the desired systems are compromised, the focus shifts to identifying, gathering, and staging valuable intellectual property or operational data. Collection consists of techniques used to aggregate data from a target network prior to exfiltration [15]. On *Linux* systems, adversaries execute this phase by staging and compressing local files into concealed archives and deploying in-memory hooks to capture sensitive user input in real time.

2.15.1. Data Staging and Archiving

Before exfiltrating data, adversaries often stage the collected information in centralized, hidden directories and compress it to reduce network footprint. For instance, operators behind the Ebury malware utilize standard *Linux* archiving utilities such as `tar` to bundle and compress harvested data into single archive files, preparing them for stealthy extraction [36].

2.15.2. Automated Input Capture

Rather than simply searching the disk, some advanced malware collects data by intercepting it in real-time as it is processed by the host. Ebury-associated implants (such as *HelimodSteal* and *FrizzySteal*) inject into applications relying on the `libcurl` library to intercept sensitive `HTTP POST` payloads. This allows the attackers to collect credit card details and login credentials submitted via online forms directly from memory, capturing the data before it is encrypted for legitimate network transit [36].

2.16. Exfiltration

After the targeted data is packaged and staged, it must be transmitted out of the victim's environment to an external server. Exfiltration consists of techniques that result or aid in an attacker removing data from a target network [15]. In *Linux* environments, threat actors achieve this by routing stolen information through established C2 frameworks, abusing legitimate cloud utilities, and tunneling data over covert proxy networks and alternative protocols.

2.16.1. Exfiltration via C2 Channels

Many malware families rely on their established C2 infrastructure to extract stolen files, blending data theft with standard beaconing traffic. Implants such as AutoColor and Linodas feature built-in functionalities to upload files directly from the infected machine to their command servers [21], [22]. Similarly, the Gomir backdoor, used by *Springtail*, and SprySOCKS, deployed by the *Earth Lusca* group, possesses dedicated API functions explicitly designed to exfiltrate arbitrary files from compromised hosts back to attacker-controlled infrastructure [19], [25].

2.16.2. Abuse of Legitimate Cloud and Web Utilities

To blend exfiltration traffic with normal administrative network activity, attackers increasingly exploit standard system utilities and cloud storage services [29]. The `curl` utility is frequently abused to transmit collected system information via `HTTP POST` requests to external infrastructure [38]. In more advanced campaigns, operators of the Interlock ransomware use enterprise tools like *Azure Storage Explorer* and *AZCopy* to silently exfiltrate massive volumes of sensitive data directly to an attacker-controlled *Azure* storage bucket [29].

2.16.3. Covert and Obfuscated Channels

To bypass strict firewall rules and network monitoring, adversaries employ alternative protocols and encrypted proxy networks to exfiltrate data. The Ebury botnet bypasses traditional network inspection by encrypting stolen credentials with an *RSA-2048* public key and transmitting them via UDP on port 53, effectively masking the exfiltration as standard DNS traffic destined for algorithmically gen-

erated domains [36]. Furthermore, to mask the destination of the stolen data, the perfctl malware routes its exfiltration traffic through the *Tor* network, uploading data to hidden Data Leak Sites (DLSs) [39]. Similarly, the CRON#TRAP campaign used no-cost, file-sharing services, such as `free.keep.sh`, providing a covert, HTTPS-encrypted exfiltration channel for SSH keys and sensitive files [32].

2.17. Impact

With the data stolen or the network fully compromised, attackers execute their final technical payload, which may disrupt, encrypt, or hijack the host's resources. Impact consists of techniques aimed at manipulating, interrupting or destroying the target system or data [15]. On *Linux* systems, adversaries inflict this final impact by deploying targeted ransomware, launching distributed Denial of Service (DOS) attacks, manipulating live network traffic, and hijacking computational resources for cryptocurrency mining.

2.17.1. Data and System Destruction

Adversaries use various methods to compromise the integrity and availability of data and systems. Ransomware deployment directly causes impact by encrypting files and systems, making them inaccessible. Ransomware groups, including *RTM Locker*, *Royal*, *Cl0p*, *BlackCat* and *GwisinLocker*, deploy variants specifically targeting *Linux*, Network Attached Storage (NAS) and ESXi virtualization systems [20], [29], [35], [52]. *RTM Locker* uses a combination of *ECDH* on *Curve25519* and *Chacha20* to encrypt files, rendering them impossible to decrypt without the attacker's private key [20]. Ransomware may perform specialized tasks, such as killing ESXi virtual machines using commands like `esxcli vm process kill` to ensure files can be encrypted [20], [35], [52]. Some ransomware samples focus solely on information theft for double extortion (charging once to decrypt data and a second time to prevent its public release), sometimes skipping the encryption phase entirely [35].

2.17.2. Denial of Service

Rather than destroying local data, some adversaries use compromised *Linux* infrastructure to disrupt external networks. Malware such as KAIJI, ShellBot and Tsunami are designed with DDoS capabilities, aiming to interrupt or deny service to a target through SYN, UDP, TCP and HTTP flood attacks [33], [47], [51].

2.17.3. Financial and Data Manipulation

The FASTCash malware manipulates *ISO8583* transaction messages to fraudulently add funds (e.g., Turkish Lira) to authorization response messages, affecting financial data integrity [18]. The *Ebury*-related tool *KernelRedirect* can modify the `Location` header of HTTP replies to redirect unsuspecting traffic to another website [36]. Furthermore, the Ebury botnet uses compromised infrastructure to cause financial and operational impact through cryptocurrency theft and large-scale spam distribution [36].

2.17.4. Resource Hijacking

Malicious actors deploy cryptomining software such as XMRig, as seen in the RUDEDEVIL and perfctl campaigns, to exploit system resources (high CPU usage) for their own benefit, significantly impacting system performance and availability [27], [33], [39]. The *8220 Gang* primarily targets cryptojacking on *Linux* and *Windows* platforms [27].

2.18. Objectives

Ultimately, attackers disrupt systems to achieve their true strategic or financial goals. That final phase consists of the socio-technical objectives of an attack that are intended to achieve a strategic goal by targeting certain assets [15]. For *Linux* threat actors, these objectives, whether financial gain via cryptomining, extortion via ransomware, or disruption via DDoS, manifest directly through the technical damage they inflict on the host. These strategic goals are covered in Sec. 2.17.

3. Conclusion

The current state-of-the-art in *Linux* offensive operations shows highly sophisticated and diverse threat landscape. As of 2026, *Linux* powers the vast majority of web servers and cloud infrastructure, making it a primary target for actors seeking resource hijacking, data exfiltration and the establishment of resilient command structures.

This report highlights several critical trends and persistent challenges defining this landscape:

- Infrastructure and C2 Innovations: attackers are shifting away from traditional dedicated servers in favor of serverless or cloud-based C2 infrastructure. By storing encrypted configuration data on public platforms like *GitHub* or *GitLab*, malicious traffic can more easily blend into legitimate network activity.
- Container and Supply Chain Risks: the widespread adoption of containerization has introduced a significant attack surface through vulnerable base images. Many of the most popular official images on *DockerHub* contain known vulnerabilities, frequently because they are built upon outdated base images.
- Operational Adaptability: offensive actors exhibit remarkable portability by using static linking to overcome architectural diversity and manipulating ELF headers to crash analysis tools like GDB. Furthermore, attackers actively abuse native *Linux* utilities to hide their malicious activities within legitimate system processes.
- Observation on the UKC: applying the 18-phase UKC framework to modern *Linux* malware highlights a tactical overlap in contemporary offensive operations. Because advanced malware, such as Ebury or perfctl, are highly modular, they frequently use the exact same underlying mechanisms (e.g., shared library hooking, eBPF manipulation or process masquerading) to simultaneously achieve Persistence, Defense Evasion and Lateral Movement. Therefore, analyzing these threats through a strict phase-by-phase chronological model generates redundancy. This overlap shows that adversaries are shifting away from fragmented, single-purpose tools in favor of consolidated, multi-purpose frameworks that collapse the boundaries between traditional kill chain phases.

Finally, it must be noted that while this report focuses exclusively on human-developed malware and traditional threat frameworks. The emerging role of Artificial Intelligence in automating offensive payload generation and adaptive evasion represents a rapidly accelerating variable. The execution of autonomous, AI-driven malware falls outside the scope of this report, but it obviously alters the trajectory of *Linux* offensive operations and must be the object of future research.

Bibliography

[1] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 161–175. DOI: 10.1109/SP.2018.00054.

[2] W3Techs, *Usage of Linux for websites*, Accessed: 2026-06-05, 2024. [Online]. Available: https://w3techs.com/technologies/details/os-linux.

[3] *Cloud Provider Linux Usage Breakdown Statistics 2026 — commandlinux.com*, https://commandlinux.com/statistics/cloud-provider-linux-usage-breakdown/, [Accessed 05-06-2026].

[4] Trend Micro, *The linux threat landscape report*, Accessed: 2024-09-16, 2023. [Online]. Available: https://www.trendmicro.com/vinfo/be/security/news/cybercrime-and-digital-threats/the-linux-threat-landscape-report.

[5] *HeroDevs Blog | Apache Struts Vulnerabilities in 2026: Critical CVEs Still Unpatched — herodevs.com*, https://www.herodevs.com/blog-posts/apache-struts-vulnerabilities-in-2026-critical-cves-still-unpatched, [Accessed 09-09-2026].

[6] S. Markovic, *32% of top-exploited vulnerabilities are over a decade old - Help Net Security — helpnetsecurity.com*, https://www.helpnetsecurity.com/2026/03/24/enterprise-vulnerability-exploitation-cybersecurity-threats/, [Accessed 09-09-2026].

[7] *ELISA – Enabling Linux In Safety Critical Applications — elisa.tech*, https://elisa.tech/, [Accessed 05-06-2026].

[8] M. Antonakakis *et al.*, "Understanding the mirai botnet," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17, Vancouver, BC, Canada: USENIX Association, 2017, pp. 1093–1110, ISBN: 9781931971409.

[9] Doctor Web. "Trojan.siggen28.58279." Added to Dr.Web virus database: 2024-11-13. Accessed: 2025-11-13. [Online]. Available: https://vms.drweb.fr/virus/?i=29159718.

[10] L. Rice, *Learning eBPF*, en. Sebastopol, CA: O'Reilly Media, Mar. 2023.

[11] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture," in *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, ser. USENIX'93 San Diego, California: USENIX Association, 1993, p. 2.

[12] *Home | tcpdump & libpcap — tcpdump.org*, https://www.tcpdump.org/, [Accessed 19-05-2025].

[13] *CVE - Search Results — cve.mitre.org*, https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=bpf, [Accessed 13-06-2026].

[14] *CVE - Search Results — cve.mitre.org*, https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=ebpf, [Accessed 13-06-2026].

[15] P. Pols and F. Domínguez, "The unified kill chain," Unified Kill Chain, White Paper, 2021, Originally published in the author's Executive Master's thesis, Cyber Security Academy, 2021. [Online]. Available: https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf.

[16] *SI110: Network Reconnaissance Lab — usna.edu*, https://www.usna.edu/Users/cs/wcbrown/courses/si110AY13S/lec/l33/lec.html, [Accessed 13-11-2025].

[17] *What is passive reconnaissance? – TechTarget Definition — techtarget.com*, https://www.techtarget.com/whatis/definition/passive-reconnaissance, [Accessed 13-11-2025].

[18] *FASTCash for Linux — haxrob.net*, https://haxrob.net/fastcash-for-linux/, [Accessed 13-11-2025].

[19] *Earth Lusca Employs New Linux Backdoor, Uses Cobalt Strike for Lateral Movement — trendmicro.com*, https://www.trendmicro.com/en_us/research/23/i/earth-lusca-employs-new-linux-backdoor.html, [Accessed 13-11-2025].

[20] *RTM Locker Ransomware as a Service (RaaS) Now on Linux - Uptycs — uptycs.com*, https://www.uptycs.com/blog/threat-research-report-team/rtm-locker-ransomware-as-a-service-raas-linux, [Accessed 13-11-2025].

[21] *Malware Spotlight: Linodas aka DinodasRAT for Linux - Check Point Research — research.checkpoint.com*, https://research.checkpoint.com/2024/29676/, [Accessed 13-11-2025].

[22] A. Armstrong, *Auto-Color: An Emerging and Evasive Linux Backdoor — unit42.paloalt*
https://unit42.paloaltonetworks.com/new-linux-backdoor-auto-color/, [Accessed 13-11-2025].

[23] M. Ezzat, *Auto-color - Linux backdoor — zw01f.github.io*, https://zw01f.github.io/malware%20analysis/auto-color/, [Accessed 13-11-2025].

[24] *Mélofée: A new alien malware in the Panda's toolset targeting Linux hosts — blog.exatrack.com*, https://blog.exatrack.com/melofee/, [Accessed 13-11-2025].

[25] *Springtail: New Linux Backdoor Added to Toolkit — security.com*, https://www.security.com/threat-intelligence/springtail-kimsuky-backdoor-espionage, [Accessed 13-11-2025].

[26] https://www.group-ib.com/blog/krasue-rat/, [Accessed 13-11-2025].

[27] *8220 Gang Cyber Threats: Cloud Infrastructure & Cryptomining Tactics — uptycs.com*, https://www.uptycs.com/blog/threat-research-report-team/8220-gang-cryptomining-cloud-based-infrastructure-cyber-threat, [Accessed 13-11-2025].

[28] *Iron Tiger's SysUpdate Reappears, Adds Linux Targeting — trendmicro.com*, https://www.trendmicro.com/en_us/research/23/c/iron-tiger-sysupdate-adds-linux-targeting.html, [Accessed 13-11-2025].

[29] S. TDR, *Interlock ransomware evolving under the radar — blog.sekoia.io*, https://blog.sekoia.io/interlock-ransomware-evolving-under-the-radar/, [Accessed 13-11-2025].

[30] *SideCopy's Multi-platform Onslaught: Leveraging WinRAR Zero-Day and Linux Variant of Ares RAT | Seqrite — seqrite.com*, https://www.seqrite.com/blog/sidecopys-multi-platform-onslaught-leveraging-winrar-zero-day-and-linux-variant-of-ares-rat/, [Accessed 13-11-2025].

[31] *APT36-Linked ClickFix Campaign Spoofs Indian Ministry of Defence, Targets Windows & Linux Users — hunt.io*, https://hunt.io/blog/apt36-clickfix-campaign-indian-ministry-of-defence, [Accessed 13-11-2025].

[32] *CRON#TRAP: Emulated Linux Environments as the Latest Tactic in Malware Staging — securonix.com*, https://www.securonix.com/blog/crontrap-emulated-linux-environments-as-the-latest-tactic-in-malware-staging/, [Accessed 13-11-2025].

[33] *Betting on Bots: Investigating Linux malware, crypto mining, and gambling API abuse — Elastic Security Labs — elastic.co*, https://www.elastic.co/security-labs/betting-on-bots, [Accessed 13-11-2025].

[34] Pangu Lab and Cyber Security Team, "The bvp47, a top-tier backdoor of us nsa equation group," Pangu Lab, Technical Report, 2022, Published February 2022. Available in English. [Online]. Available: https://www.pangulab.cn/files/The_Bvp47_a_top-tier_backdoor_of_us_nsa_equation_group.en.pdf.

[35] M. S. Fernandez, *The Platform Matters: A Comparative Study on Linux and Windows Ransomware Attacks - Check Point Research — research.checkpoint.com*, https://research.checkpoint.com/2023/the-platform-matters-a-comparative-study-on-linux-and-windows-ransomware-attacks/, [Accessed 13-11-2025].

[36] M.-E. M. Léveillé, "Ebury is alive but unseen: 400k Linux servers compromised for cryptocurrency theft and financial gain," ESET Research, White Paper, May 2024, Accessed on October 28, 2026. [Online]. Available: https://web-assets.esetstatic.com/wls/en/papers/white-papers/ebury-is-alive-but-unseen.pdf.

[37] Z. Reichert, *Unveiling — aon.com*, https://www.aon.com/en/insights/cyber-labs/unveiling-sedexp, [Accessed 13-11-2025].

[38] *How Malicious Actors Abuse Native Linux Tools in Their Attacks — trendmicro.com*, https://www.trendmicro.com/en_us/research/22/i/how-malicious-actors-abuse-native-linux-tools-in-their-attacks.html, [Accessed 13-11-2025].

[39] *Perfctl: A Stealthy Malware Targeting Millions of Linux Servers — aquasec.com*, https://www.aquasec.com/blog/perfctl-a-stealthy-malware-targeting-millions-of-linux-servers/, [Accessed 13-11-2025].

[40] F. FKIE, *SideCopy (Threat Actor) — malpedia.caad.fkie.fraunhofer.de*, https://malpedia.caad.fkie.fraunhofer.de/actor/sidecopy, [Accessed 13-06-2026].

[41] J. Moore, *"Shai-Hulud" Worm Compromises npm Ecosystem in Supply Chain Attack (Updated November 26) — unit42.paloaltonetworks.com*, https://unit42.paloaltonetworks.com/npm-supply-chain-attack/, [Accessed 13-06-2026].

[42] H. Chen, Y. Mao, X. Wang, D. Zhou, N. Zeldovich, and M. F. Kaashoek, "Linux kernel vulnerabilities: State-of-the-art defenses and open problems," in *Proceedings of the 6th Asia-Pacific Workshop on Systems (APSys '11)*, ACM, 2011. DOI: 10.1145/2103799.2103804. [Online]. Available: https://dblp.org/rec/conf/apsys/ChenMWZZK11.

[43] *School of Computer Science - University of Birmingham — cs.bham.ac.uk*, https://www.cs.bham.ac.uk/~exr/lectures/opsys/12_13/docs/kernelAPI/r3859.html, [Accessed 13-11-2025].

[44] *Null Dereference | OWASP Foundation — owasp.org*, https://owasp.org/www-community/vulnerabilities/Null_Dereference, [Accessed 10-06-2026].

[45] *Sysrv Infection (Linux Edition) | dfir.ch — dfir.ch*, https://dfir.ch/posts/sysrv/, [Accessed 13-11-2025].

[46] *Linux Detection Engineering - The Grand Finale on Linux Persistence — Elastic Security Labs — elastic.co*, https://www.elastic.co/security-labs/the-grand-finale-on-linux-persistence, [Accessed 13-11-2025].

[47] https://asec.ahnlab.com/en/54647/, [Accessed 13-11-2025].

[48] J. Baines, *Programming Linux Anti-Reversing Techniques*. Syngress, 2016, ISBN: 978-0128033010.

[49] *Symbiote: A New, Nearly-Impossible-to-Detect Linux Threat — intezer.com*, https://intezer.com/blog/new-linux-threat-symbiote, [Accessed 13-06-2026].

[50] *Chinese hackers use new Linux malware variants for espionage — bleepingcomputer.com*, https://www.bleepingcomputer.com/news/security/chinese-hackers-use-new-linux-malware-variants-for-espionage/, [Accessed 13-11-2025].

[51] https://asec.ahnlab.com/en/49769/, [Accessed 13-11-2025].

[52] *Royal Ransomware Expands Attacks by Targeting Linux ESXi Servers — trendmicro.com*, https://www.trendmicro.com/en_us/research/23/b/royal-ransomware-expands-attacks-by-targeting-linux-esxi-servers.html, [Accessed 13-11-2025].

Glossary

Auto-Color An evasive *Linux* backdoor that grants threat actors persistent remote access by leveraging custom encryption and an `LD_PRELOAD`-based library implant to hide its network connections and protect itself from removal. 14, 25, 29, 32, 41, 46, 48

Bvp47 A sophisticated, top-tier *Linux* backdoor attributed to the *NSA*-linked Equation Group that leverages a custom BPF covert communication channel triggered by TCP SYN packets, extensive kernel-level system hooking, modular encrypted payloads and host-bound environment verification to maintain long-term, covert remote control. 32, 46

ClickFix A social engineering attack technique and infection method that uses cloned, trusted web interfaces (such as fake verification or CAPTCHA prompts) to silently copy malicious commands to a victim's clipboard and trick them into manually pasting and executing them in their terminal or *Run* dialog to deliver cross-platform malware payloads. i, 14, 16, 18, 41

CRON#TRAP An attack campaign and malware staging tactic delivered via phishing (`.lnk` files) that deploys a lightweight, custom *Tiny Core Linux* environment emulated through QEMU to maintain a stealthy endpoint presence and establish persistent backdoor access to an attacker-controlled C2 server. 41, 46, 49

Ebury An advanced *OpenSSH* backdoor and credential-stealing *Trojan* targeting *Linux* that patches *OpenSSH* binaries or libraries to intercept login credentials and private SSH keys, facilitate remote access and enable secondary operations such as spam campaigns, traffic redirection and cryptocurrency theft. 22, 24, 25, 29, 43, 44, 46–48, 51, 53

FASTCash A North Korean *Linux*-variant (after *IBM AIX* and *Windows*) malware installed on network payment switches to enable unauthorized cash withdrawals from ATMs. 14, 51

Gh0st An open-source RAT serving as the indirect inspiration for Linodas C2 functionality. 14

Gomir A *Go*-based backdoor deployed via trojanized software packages that establishes persistent *Linux* access (using `systemd` or `cron`) to execute arbitrary shell commands, manage reverse proxying and exfiltrate sensitive files. 14, 41, 48

IceFire A ransomware family that expanded its operations to target *Linux* environments, known for exploiting vulnerabilities in enterprise-facing applications (such as *IBM* technologies) to gain initial access, encrypt critical files and extort victim organizations. 21

Interlock A ransomware intrusion set first observed in September 2024 across North America and Europe leveraging multi-stage infection chains with fake software updaters, social engineering techniques and backdoors to deploy its payload and leak stolen victim data. 16, 48

KAIJI A *Linux* malware family known for its distributed DDoS capabilities and flooder functionality, actively maintained and deployed by threat actors to exploit system resources alongside cryptocurrency miners following web server compromises. 24, 46, 50

Krasue A *Linux* RAT primarily targeting Thai organizations using embedded open-source LKM rootkits to hook syscalls and disguise its presence, while employing RTSP messages for stealthy C2 alive pings. 14, 25, 39

Linodas Also known as *XDealer*, it is a multi-platform remote access trojan linked to Chinese cyberespionage actors that targets *Windows* and *Linux* servers, providing persistent backdoor access, extensive system enumeration, reverse shell capabilities and defense evasion through system binary proxying. 14, 23, 32, 48, 60

Mélofée An elusive *Linux* malware family linked to Chinese state-sponsored threat groups that features modular backdoors, customizable network protocols and an integrated kernel-mode rootkit for stealth and persistence. 14, 25, 46

perfctl A highly evasive Linux malware targeting misconfigured and vulnerable servers to deploy cryptocurrency miners and backdoors, utilizing rootkits, process masquerading, *TOR*-based external communications and dormant states during active user sessions to remain undetected. 17, 21, 23, 24, 29, 35, 39, 44, 49, 51, 53

Royal A ransomware strain operated by seasoned cybercriminals formerly associated with *Conti* that has expanded from *Windows* to target *Linux* ESXi servers. 39

RTM Locker A Ransomware-as-a-Service (RaaS) strain inspired by the *Babuk* group's leaked source code that targets *Linux*, NAS and ESXi environments, utilizing specific `esxcli` commands to terminate virtual machines and combining specific encryption schemes to lock victim files. 14, 39, 42

RUDEDEVIL A *Linux* malware strain deployed on compromised web servers to exploit system resources by contacting its C2 server, retrieving configuration parameters and downloading and executing the XMRig binary for cryptocurrency mining. 51, 63

sedexp A stealthy *Linux* malware used by financially motivated threat actors that ensures reboot persistence, provides reverse shell capabilities and uses memory manipulation to conceal files and configurations from system commands. 25, 32, 38

ShellBot A *Perl*-based DDoS bot malware deployed against poorly managed *Linux* SSH servers compromised via dictionary and brute-force attacks, utilizing the Internet Relay Chat (IRC) protocol to communicate with its C2 server and receive attack commands. 32, 45, 50

SideCopy A Pakistan-linked APT group targeting Indian and Afghan defense and government entities that conducts multi-platform cyberespionage campaigns by exploiting vulnerabilities like `WinRAR` to deliver payloads and Linux variants of the open-source `Ares` RAT. 46

SimpleRemoter An open-source remote access tool based on RAT including C2 command functionality, OS version detection and `zlib` library usage. 14

SprySOCKS A Linux backdoor derived from a *Windows* malware named *Trochilus* and utilized by the *Earth Lusca* threat group, featuring SOCKS proxy capabilities, an interactive shell, file manipulation and system reconnaissance functions to facilitate cyberespionage. 14, 23, 48

Symbiote A highly evasive Linux malware deployed as a shared object library loaded via `LD_PRELOAD` into running processes to provide rootkit functionality, harvest credentials and grant backdoor access while utilizing BPF hooking and `libc` function interception to conceal its files, processes and network traffic. 29

Sysrv A *Go*-based worm and cryptomining botnet targeting *Linux* and *Windows* systems that propagates by scanning the internet, exploiting unpatched vulnerabilities in web applications and exposed services, and deploying automated scripts to download and execute cryptocurrency miners. 21

SysUpdate A custom malware family used by the *Iron Tiger* advanced APT group that features dynamic libraries loading, process hollowing and multi-stage payload execution, which has been expanded from *Windows* to compromise *Linux* systems. 23, 32

Tsunami A *Linux* malware strain deployed alongside a cryptocurrency miner in *8220 Gang*'s campaigns against cloud-based infrastructure. 24, 35, 41, 50

XMRig An open-source cryptocurrency mining binary that is downloaded, configured and executed by malware variants (such as RUDEDEVIL) on compromised *Linux* servers to mine cryptocurrency utilizing system resources. 51, 62

Acronyms

ABI Application Binary Interface. 2, 4

ACL Access Control List. 12

AitM Adversary-in-the-Middle. iii, 34, 43, 46

API Application Programming Interface. 26, 28, 48

APT Advanced Persistent Threat. 7, 9, 11, 30, 62

BPF Berkeley Packet Filter. 7–9, 30, 32, 60, 62

BSD Berkeley Software Distribution. 8

C2 Command & Control. i–iii, 1, 7, 9, 10, 14, 29, 31–33, 48, 53, 60–62

CD Continuous Deployment. 17

CDN Content Delivery Network. 10

CI Continuous Integration. 17

CKC Cyber Kill Chain. 11

CVE Common Vulnerabilities and Exposure. 9

CWP Control Web Panel. 21

DDoS Distributed Denial of Service. 3, 6, 50, 52, 61, 62

DLS Data Leak Site. 49

DNS Domain Name System. 12, 13, 29, 32, 35, 48

DOS Denial of Service. 50

eBPF extended Berkeley Packet Filter. i, ii, 1, 7–9, 30, 53

ELF Executable and Linkable Format. 3, 4, 15, 16, 22, 24, 26, 53

ESXi Elastic Sky X integrated. 40, 50, 61

FS FileSystem. 22, 23, 29, 35

GDB GNU Debugger. 4, 28, 53

GRUB Grand Unified Bootloader. 23, 39

initramfs Initial RAM FileSystem. 23, 39

IoT Internet of Things. 2, 3, 6

IPC Inter-Process Communication. 32, 33

IRC Internet Relay Chat. 62

KCP Keep Connection Protocol. 31

LKM Loadable Kernel Module. 9, 30, 61

NAS Network Attached Storage. 50, 61

OS Operating System. i, 1–5, 12, 32, 46

OSINT Open source intelligence. 13, 36

OWASP Open Worldwide Application Security Project. 1, 2, 20

QEMU Quick Emulator. 16, 41, 60

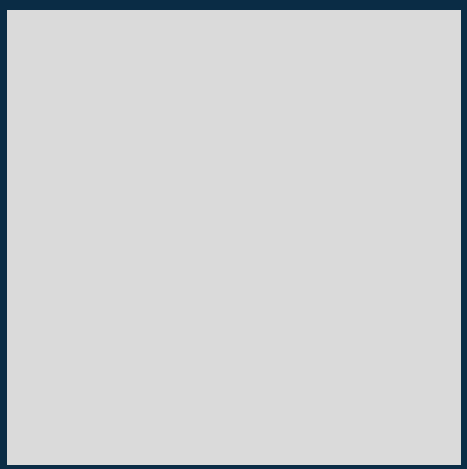
RaaS Ransomware-as-a-Service. 61

RAT Remote Access Trojan. 6, 7, 14, 46, 60–62

RCE Remote Code Execution. 21

RTSP Real Time Streaming Protocol. 32

UAC User Account Control. 28

UKC Unified Kill Chain. 11, 16, 53

UPX Ultimate Packer for Executables. 26