REPORT 4 : ULOGD

CONTENT

I. INTRODUCTION

In modern network systems, logging and analyzing network traffic is a crucial factor in ensuring security and managing communication activities. ULOGD (Userspace Logging Daemon) is a userspace logging daemon designed to capture packets and traffic data from the Netfilter/Iptables firewall system.

ULOGD supports per-packet logging, per-flow logging, as well as flexible and user-defined accounting modes. Its architecture is based on a plugin system with three main components: Input plugins (to collect data from Netfilter), Filter plugins (to process, analyze, and transform the data), and Output plugins (to write the data to specific log formats such as text files, databases, or JSON).

Additionally, ULOGD allows for configuration using a stack concept—a chain of plugins linked in sequence to form a flexible and efficient data processing path. Integration with plugins such as ulogd_output_LOGEMU, ulogd_output_JSON, and ulogd_output_MYSQL enables the system to log data in various formats for monitoring and analysis.

This report presents key concepts, plugin structure, configuration methods, and practical examples of using ULOGD to log network data on Linux systems, thereby supporting effective network operation and security in organizations.

II. PREREQUISITES

To effectively work with ULOGD in the context of network traffic logging and analysis, it is essential to understand some foundational concepts and prepare the necessary hardware and software components. This section outlines the required knowledge, theoretical background, hardware setup, and software tools needed before implementation.

1. Required Knowledge & Theory

1.1 Overview

ULOGD is a userspace logging daemon used for logging events related to Netfilter/Iptables in Linux. It enables various types of network logging, including:

- Per-packet logging for identifying security violations or monitoring specific traffic.

- Per-flow logging for accounting and performance analysis,
- Flexible, user-defined logging pipelines tailored to specific needs.

ULOGD plays a critical role in translating raw packet data into structured log outputs through a plugin-based architecture.

1.2 Stack Concept

ULOGD uses a stack-based plugin system, where each plugin performs a specific function within the logging pipeline. Each plugin is defined by:

- Input Keys: Data items required by the plugin to function.
- Output Keys: The resulting key-value pairs the plugin produces after processing.

Plugins are connected in a logical sequence (stack), with output from one plugin feeding into the next, enabling modular and scalable logging pipelines.

1.3 Plugin Types

ULOGD 2.x is designed to be highly modular and extensible. It comprises a lightweight core and a wide variety of plugins. The system is categorized into three main plugin types:

- Input Plugins: These act as data sources, retrieving packets or flow information.
    - ULOG: Captures packets from the legacy iptables ULOG target.
    - NFLOG: Captures packets via the modern NFLOG interface (recommended).
    - NFCT: Retrieves flow data via Netfilter connection tracking.
- Filter Plugins: Process and transform raw packet data into structured formats.
    - Example: Parsing raw packets into fields like IP addresses, protocols, or ports.
- Output Plugins: Define how and where the processed data is saved.
    - Examples include:

      LOGEMU: Logs to a plain text file.

      SYSLOG: Sends logs to the system logger.

      MYSQL, PGSQL, SQLITE3: Logs to various SQL databases.

      JSON: Outputs logs in structured JSON format.

      PCAP, IPFIX: For binary and flow-export use cases.

These plugins are chained together into a logging stack, allowing users to create powerful and flexible logging workflows.

2. Hardware Requirements

To implement ULOGD in a practical environment, the following hardware is required:

- 1 Raspberry Pi running a compatible Linux-based OS (e.g., WebOS image).
- Ethernet cables (pre-crimped) for connecting the Raspberry Pi to an isolated network.

3. Software Tools

Ensure the following software components are installed and properly configured:

- ULOGD v2.0.7: The core logging daemon with required plugins.
- iptables: To redirect traffic to the NFLOG target for packet capturing.
- logrotate: For automatic management and rotation of log files.
- vim/nano: Text editors used to edit ULOGD configuration files (e.g., /etc/ulogd.conf).

III. RESEARCH ABOUT PLUGINS

Ulogd2 (Netfilter Userspace Logging Daemon) is a background service that enables flexible logging of firewall events from Netfilter/iptables. It supports both per-packet logging (e.g., for policy violations or attacks) and connection-based logging (for auditing and traffic statistics). ulogd2 uses a plugin-based architecture: a small core integrates multiple plugins responsible for input (receiving data from the kernel), filters (processing/extracting information), and output (writing logs to the desired destination).

Thanks to structured outputs (JSON, XML, databases), logs from IoT devices can be sent to a central server or integrated into a SIEM system for anomaly analysis.

1. INPUT PLUGINS

Input plugins serve as the data sources for ulogd2—they receive information from the kernel (Netfilter) and convert it into key-value fields for ulogd2. Each input plugin handles different types of events in Netfilter :

- **ULOG:** Receives packets from the iptables **ULOG** target. This is the older logging method (Linux kernel 2.4.x), where iptables sends packets to a userspace queue. The ULOG plugin listens on the specified netlink group and retrieves these packets.(Note: ULOG is primarily for IPv4 and has been replaced by NFLOG on newer kernels)
- **NFLOG**: Receives packets from the **NFLOG** target (via **nfnetlink_log)**—the modern, recommended mechanism for logging (kernel 2.6.14+). NFLOG supports logging for all protocol families (IPv4, IPv6, bridging, etc.). The kernel sends matched packets to netlink (in a defined group); the NFLOG plugin opens a netlink socket to receive them.

Advantage: Improved performance—multiple packets can be grouped into one message to reduce context-switching between kernel and user space (configurable via **--nflog-qthreshold**)

- **NFCT** – Receives connection tracking events from Netfilter's conntrack system. The NFCT plugin (Netfilter ConnTrack) uses **libnetfilter_conntrack** to monitor events like new or closed TCP/UDP connections. It provides **flow-based logs** instead of per-packet, including source/destination IP, ports, protocols, timestamps, bytes/packets transmitted, etc.—ideal for traffic auditing.Admins can configure NFCT to log only on session close (to get full stats) or on all events. Flow-based logging significantly reduces log volume, making it suitable for resource-limited IoT devices.

2. OUTPUT PLUGINS

Output plugins determine how and where the log information is written. ulogd2 supports various output types to meet different needs—from simple text files to structured formats like pcap, databases, or JSON/XML for analysis.

- **LOGEMU** – Logs to a text file in a format that mimics traditional kernel logs. Each packet/flow is written on one line with a familiar iptables-like syntax (timestamp, interface, IPs, packet info…). Useful for separating firewall logs into dedicated files like **/var/log/syslogemu.log.**
- **OPRINT** – Logs to a text file in a multiline format. Unlike LOGEMU (one concise line), OPRINT can split log entries into multiple lines for each field—useful for structured readability (e.g., start/end time, bytes sent/received on separate lines).
- **SYSLOG** – Sends logs to the system's syslog. This plugin uses libc's syslog interface to integrate ulogd2 logs into **/var/log/syslog** or journald, like other system services. Note: On WebOS systems, this plugin may not be enabled by default, as logs are often handled via JSON/XML or the system journal.
- **MySQL / PostgreSQL / SQLite3** – These plugins log data into SQL databases (MySQL, PostgreSQL, or SQLite3). Each corresponding plugin inserts log entries into a specific table in the database, usually via predefined stored procedures. For example, **ulogd2** provides sample schemas such as the **ulog** table for packet logs and the **ulog2_ct** table for connection logs, along with procedures like **INSERT_PACKET_FULL** and **INSERT_CT** to insert records.Administrators need to configure the connection details

(host, user, password, database name, table name, etc.) in the **ulogd2** configuration file. **Advantages**: Logs stored in SQL databases can be easily queried, analyzed, and integrated with monitoring applications (e.g., the Nulog web interface for viewing firewall logs). **Disadvantages**: Logging to databases can be relatively slow, so it is typically used only for important events or on sufficiently powerful systems.

- **PCAP** – This plugin writes packets to **.pcap** files. The PCAP plugin allows logging of entire packet contents (similar to using **tcpdump**) into a **.pcap** file. These files can then be opened using Wireshark or other packet analysis tools to inspect details like payloads and headers.The PCAP plugin is commonly used with the **NFLOG** or **ULOG** input plugins, as they provide raw packet data.For example, configuring the stack **log2:NFLOG,...,pcap1:PCAP** will save all packets matched by the **NFLOG** rule in group 2 into a .pcap file (e.g., /var/log/ulogd.pcap).This feature is powerful for investigating suspicious packets on IoT devices (e.g., malware detection, analyzing unknown protocols), but care must be taken as .pcap files can grow very large.
- **IPFIX** – This plugin exports logs in the IP Flow Information Export (IPFIX) format. IPFIX is a standard protocol used to transmit flow records to centralized collection systems.The IPFIX plugin in ulogd2 acts as an exporter, periodically sending flow records (collected from plugins like NFCT or NFACCT) to a collector server using the IPFIX format (usually over UDP).Administrators can configure parameters such as the collector host, port, and template fields.IPFIX is especially useful in IoT environments where data from multiple devices must be gathered and analyzed on a central monitoring system using a standard protocol.
- **NACCT** – This plugin logs network traffic statistics in the Netfilter Accounting format (compatible with the legacy nacct tool). It is typically used with the NFCT or NFACCT input plugins to export traffic statistics to a file in a simple format (e.g., each line contains timestamp, source/destination IP addresses, bytes, packets, etc.).The “nacct compatible” format means the log file is structured so that legacy scripts or tools from the old Netfilter toolkit can directly parse it to generate reports.In other words, the NACCT plugin provides a traditional text-based accounting log that is easy to read or process with lightweight tools, compared to more complex formats like IPFIX or SQL.Administrators

can specify the file path where logs are written, and the plugin will add new records each time an event (e.g., a connection ends) occurs, including the associated traffic statistics.

- **JSON** –This plugin writes logs in **JSON (JavaScript Object Notation)** format. Introduced in ulogd2 version 2.0.4, the JSON output plugin records each log event as a key-value JSON object in a file (default: /var/log/ulogd.json).**Advantages**: JSON logs are easy to parse and analyze with other tools—for instance, Logstash/Elasticsearch can read ulogd2 JSON logs and visualize them using Kibana.The JSON plugin also supports optional fields like numeric_label and boolean_label (from the NFLOG plugin) to indicate whether a packet was accepted or dropped.In IoT environments, JSON logging is particularly useful for sending logs to a centralized server or integrating with cloud services, since JSON is a web-standard data format.
- **XML** –This plugin writes logs in **XML** format. If enabled, the XML plugin sequentially writes log events as XML nodes into a designated file (or folder).XML provides structured formatting similar to JSON and is easily parsed by software, though it is more verbose and less commonly used due to its complexity.ulogd2 supports logging of packet events, connections, or traffic statistics in XML format.For example, configuring the stack ct1:NFCT, xml1:XML will log all connection events into structured XML elements.The XML plugin is typically used only when a log management system explicitly requires XML; otherwise, JSON is preferred due to its lightweight nature.

3. SUMMARY

Ulogd2 offers a variety of initial options to meet different logging goals. If you want to monitor directly on the device – LOGEMU or SYSLOG can be used to view logs directly in a text/syslog file. If you need long-term storage and analysis – SQL or JSON/XML databases can be used to import into system analysis. For in-depth packet analysis, PCAP can be used to store the entire packet for review. For network storage monitoring (network accounting) – formats such as IPFIX or NACCT should be used to aggregate alternative information streams for each packet. Output plugins can be combined with most input plugins, but compatibility should be noted: for example, PCAP plugins only make sense when the input provides raw packet data (NFLOG/ULOG) – not NFCT (since the connection event does not contain packets to capture). Similarly, IPFIX or NACCT output is primarily for streams (NFCT/NFACCT);

LOGEMU/OPRINT/JSON is quite flexible, being able to write either log packets or log streams (there is no compatible filter plugin format like PRINTPKT or PRINTFLOW). The merge plugin must be configured in a stack – typically each stack has 1 input, which needs some filters, and then finally 1 output. For example, a log stack packet might be NFLOG -> [IP decoding filters, interfaces] -> PRINTPKT -> LOGEMU; while the connection log stack might be NFCT -> [IP to string filter] -> PRINTFLOW -> OPRINT. If the combination is incorrect (like NFCT directly to PCAP), ulogd2 will not log due to lack of matching data.

- **ulogd2 INPUT and OUPUT compatibility table**

Input Plugin	Output Plugins Compatibility	NOTE
ULOG	**LOGEMU, OPRINT, SYSLOG, JSON, XML**	Supports packet logging in text or structured format. PCAP is not supported since ULOG does not provide full packet contents like NFLOG.
	MySQL / PostgreSQL / SQLite3	Need to add IP2STR, PRINTPKT, SQL filters for formatting.
NFLOG	**LOGEMU, OPRINT, SYSLOG, JSON, XML**	Best for packet logging. Can log text or structured JSON.
	PCAP	Record all packets in .pcap format (similar to tcpdump). Must combine appropriate PCAP filter.
	MySQL / PostgreSQL / SQLite3	Use PRINTPKT, SQL filter. Useful for storing structured packet logs.
NFCT	**OPRINT, SYSLOG, JSON, XML**	Connection/flow log – not packet log.
	MySQL / PostgreSQL / SQLite3	Very suitable for logging log flows (eg ulog2_ct table).

	IPFIX, **NACCT**	Export flows to a centralized system or save to an accounting file.
	PCAP, **LOGEMU**	Cannot be used because there is no packet data to capture/log in pcap or raw log format.

- **Table of popular plugin combinations:**

Input	Filter (decode)	Suitable output	Mục tiêu
NFLOG	**BASE, IP2STR, PRINTPKT**	**LOGEMU / OPRINT / JSON**	Log each packet in text/structure form
NFLOG	**BASE, PCAP**	**PCAP**	Capture packet contents for analysis using Wireshark
NFLOG	**BASE, PRINTPKT, SQL**	**MySQL / PostgreSQL**	Log each package into the database for statistics
NFCT	**BASE, IP2STR, PRINTFLOW**	**OPRINT / JSON / SQL**	Log full TCP/UDP streams (IP, port, bytes, time...)
NFCT	**BASE**	**IPFIX / NACCT**	Send traffic statistics to centralized system

IV. BUILD AND CONFIGURATION

1. FILES REQUIRED FOR ULOGD CONFIGURATION

To enable **ULOGD** to export logs in **JSON** format, the **libjansson.so** library is required. Therefore, the **jansson.bb** build recipe must be added, and a few modifications must be made to the **ulogd.bb** recipe.

1.1 Setting up the Jansson Library

Version **2.9** of Jansson is used. Jansson is a C library for encoding, decoding, and manipulating JSON data.

- **jansson.bb:**

```
SUMMARY = "Jansson is a C library for encoding, decoding and
manipulating JSON
data"
HOMEPAGE = "http://www.digip.org/jansson/"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM =
"file://LICENSE;md5=8b70213ec164c7bd876ec2120ba52f61"
SRC_URI =
"http://www.digip.org/jansson/releases/${BPN}-${PV}.tar.gz"
SRC_URI[md5sum] = "84abaefee9502b2f2ff394d758f160c7"
SRC_URI[sha256sum] =
"0ad0d074ca049a36637e7abef755d40849ad73e926b93914ce294927b97bd2a5"
inherit autotools pkgconfig
```

1.2 Modifying ulogd.bb

```
SUMMARY = "Netfilter userspace logging daemon"
DESCRIPTION = "Userspace logging daemon for netfilter/iptables
related logging"
HOMEPAGE = "http://www.netfilter.org/projects/ulogd/index.html"
LICENSE = "GPL-2.0-only"
LIC_FILES_CHKSUM =
"file://COPYING;md5=c93c0550bd3173f4504b2cbd8991e50b"
PV = "2.0.7"
SRC_URI = " \

http://www.netfilter.org/projects/ulogd/files/ulogd-${PV}.tar.bz2;n
ame=tar \
```

```
  file://ulogd.init \
  file://ulogd.service \
  file://ulogd.conf.in.user \
  file://ulogd.logrotate.user \
"
SRC_URI[tar.md5sum] = "2bb2868cf51acbb90c35763c9f995f31"
SRC_URI[tar.sha256sum] =
"990a05494d9c16029ba0a83f3b7294fc05c756546b8d60d1c1572dc25249a92b"

DEPENDS = "libnfnetlink libmnl libnetfilter-log
libnetfilter-conntrack jansson"
RDEPENDS_${PN} = "libnfnetlink libmnl libnetfilter-log
libnetfilter-conntrack jansson"

EXTRA_OECONF = "\
 --disable-nfacct \
 --without-dbi \
 --without-sqlite \
 --without-pgsql \
 --without-mysql \
 --without-pcap \
"
PACKAGECONFIG ??= "nfacct"

inherit autotools manpages pkgconfig systemd update-rc.d

do_install:append () {
        install -d ${D}${sysconfdir}
        install -m 0644 ${WORKDIR}/ulogd.conf.in.user
${D}${sysconfdir}/ulogd.conf
        install -d ${D}${sysconfdir}
        install -m 0644 ${WORKDIR}/ulogd.logrotate.user
${D}${sysconfdir}/ulogd.logrotate
```

```
        install -d ${D}${mandir}/man8
        install -m 0644 ${S}/ulogd.8 ${D}${mandir}/man8/ulogd.8

        install -d ${D}${systemd_system_unitdir}
        install -m 0644 ${WORKDIR}/ulogd.service
${D}${systemd_system_unitdir}
        sed -i -e 's,@SBINDIR@,${sbindir},g'
${D}${systemd_system_unitdir}/ulogd.service

       install -d ${D}${sysconfdir}/init.d
       install -m 755 ${WORKDIR}/ulogd.init
${D}${sysconfdir}/init.d/ulogd
     install -d ${D}${libdir}/ulogd

}

PACKAGES += "${PN}-plugins"
ALLOW_EMPTY:${PN}-plugins = "1"
PACKAGES_DYNAMIC += "^${PN}-plugin-.*$"
FILES_${PN} += "${libdir}/ulogd/*.so"

python split_ulogd_libs () {
    libdir = d.expand('${libdir}/ulogd')
    dbglibdir = os.path.join(libdir, '.debug')

    split_packages = do_split_packages(d, libdir,
r'^ulogd_.*\_([A-Z0-9]*).so', '${PN}-plugin-%s', 'ulogd2 %s
plugin', prepend=True)
    split_dbg_packages = do_split_packages(d, dbglibdir,
r'^ulogd_.*\_([A-Z0-9]*).so', '${PN}-plugin-%s-dbg', 'ulogd2 %s
plugin - Debugging files', prepend=True, extra_depends='${PN}-dbg')

    if split_packages:
```

```
        pn = d.getVar('PN')
        d.setVar('RRECOMMENDS:' + pn + '-plugins', '
'.join(split_packages))
        d.appendVar('RRECOMMENDS:' + pn + '-dbg', ' ' + '
'.join(split_dbg_packages))
}
PACKAGESPLITFUNCS:prepend = "split_ulogd_libs "

SYSTEMD_SERVICE:${PN} = "ulogd.service"
INITSCRIPT_NAME = "ulogd"
INITSCRIPT_PARAMS = "defaults"
```

1.3 Replace configuration text in /files/ulogd.conf.in.user:

The **ulogd.conf.in.user** file is a **customized configuration template** for **ulogd**, the userspace logging daemon for Netfilter/iptables. This file defines **how ulogd behaves**, including which plugins to load, how to process captured packets, and where to log output. Below is a breakdown of its key sections and the purpose of the modifications.

A. Global Configuration

```
# Example configuration for ulogd
# Adapted to Debian by Achilleas Kotsis <achille@debian.gr>


##############################################################
#######
# GLOBAL OPTIONS
##############################################################
#######


# logfile for status messages
[global]
logfile="/var/log/ulogd.log"
```

```
# loglevel: debug(1), info(3), notice(5), error(7) or fatal(8)
(default 5)
# loglevel=1
```

- Specifies the **log file path** for internal ulogd status messages (not packet logs).
- This helps in debugging and verifying that the ulogd daemon is running correctly.

B. Plugin Loading

The configuration explicitly lists dynamic plugins to be loaded. Each plugin extends `ulogd` to support specific input, filtering, or output capabilities.

```
#########################################################################
#######
#
# PLUGIN OPTIONS
#########################################################################
#######
# We have to configure and load all the plugins we want to use
# general rules:
# 1. load the plugins _first_ from the global section
# 2. options for each plugin in seperate section below

#plugin="/usr/lib/ulogd/ulogd_LOGEMU.so"
plugin="/usr/lib/ulogd/ulogd_inppkt_NFLOG.so"
#plugin="/usr/lib/ulogd/ulogd_inppkt_ULOG.so"
#plugin="/usr/lib/ulogd/ulogd_inppkt_UNIXSOCK.so"
plugin="/usr/lib/ulogd/ulogd_inpflow_NFCT.so"
plugin="/usr/lib/ulogd/ulogd_filter_IFINDEX.so"
plugin="/usr/lib/ulogd/ulogd_filter_IP2STR.so"
plugin="/usr/lib/ulogd/ulogd_filter_IP2BIN.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_filter_IP2HBIN.s
o"
plugin="/usr/lib/ulogd/ulogd_filter_PRINTPKT.so"
plugin="/usr/lib/ulogd/ulogd_filter_HWHDR.so"
plugin="/usr/lib/ulogd/ulogd_filter_PRINTFLOW.so"
```

```
#plugin="/usr/lib/ulogd/ulogd_filter_MARK.so"
plugin="/usr/lib/ulogd/ulogd_output_LOGEMU.so"
plugin="/usr/lib/ulogd/ulogd_output_SYSLOG.so"
plugin="/usr/lib/ulogd/ulogd_output_XML.so"
#plugin="/usr/lib/ulogd/ulogd_output_SQLITE3.so"
plugin="/usr/lib/ulogd/ulogd_output_GPRINT.so"
plugin="/usr/lib/ulogd/ulogd_output_NACCT.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_PCAP.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_PGSQL.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_MYSQL.so"
#plugin="/usr/lib/x86_64-linux-gnu/ulogd/ulogd_output_DBI.so"
plugin="/usr/lib/ulogd/ulogd_raw2packet_BASE.so"
plugin="/usr/lib/ulogd/ulogd_inpflow_NFACCT.so"
plugin="/usr/lib/ulogd/ulogd_output_GRAPHITE.so"
plugin="/usr/lib/ulogd/ulogd_output_JSON.so"
```

Plugin Categories:

- **Input plugins**: Collect packets (e.g., **NFLOG, NFCT**)
- **Filter plugins**: Extract, convert, or format data (e.g., **IP2STR, PRINTPKT**)
- **Output plugins**: Store or forward logs (e.g., **LOGEMU, JSON, SYSLOG**)

These plugins are explicitly listed rather than using default auto-discovery, giving users **fine-grained control**.

C. Stack Configuration

Input plugin: nflog0:NFLOG reads packets from NFLOG group 32.

Filter plugins:

- **BASE:** Extracts basic packet information (protocol, ports, etc.)
- **IFINDEX:** Captures network interface index
- **HWHDR:** Gets MAC address (hardware header)
- **IP2STR:** Converts IP addresses to strings
- **PRINTPKT:** Prints raw packet details in human-readable form

Output plugin

- **LOGEMU** logs this information to a plain text file /var/log/full.log.

```
stack=nflog0:NFLOG,base0:BASE,ifi0:IFINDEX,hwhdr0:HWHDR,ip2str0
:IP2STR,print1:PRINTPKT,full0:LOGEMU
```

- Same structure as stack 1, but listens to **NFLOG group 33**.
- Allows for separating logs based on traffic direction or policy (e.g., **INPUT vs OUTPUT**).

```
stack=nflog1:NFLOG,base0:BASE,ifi0:IFINDEX,hwhdr0:HWHDR,ip2str0
:IP2STR,print1:PRINTPKT,full0:LOGEMU
```

Input: nflog2:NFLOG for NFLOG group 34

Filters:

- **BASE, IFINDEX, IP2STR** — same as above
- **HWHDR** (named mac2str1) — extracts MAC address

Output: JSON plugin writes structured logs to /var/log/firewall_log.json

```
stack=nflog2:NFLOG,base0:BASE,ifi0:IFINDEX,ip2str0:IP2STR,mac2s
tr1:HWHDR,json1:JSON
```

D. NFLOG Group Mapping

Each stack listens to a specific Netfilter group:

[nflog0] → group=32

[nflog1] → group=33

[nflog2] → group=34

```
[nflog0]
group=32

[nflog1]
```

```
group=33

[nflog2]
group=34
```

E. Output configuration:

- **sync=1:** Ensures that data is immediately written to disk, minimizing log loss on system crash.
- **full.log:** Logs in plain text (LOGEMU).
- **firewall_log.json:** Logs in JSON format, suitable for use with logging systems like ELK, Splunk, etc.

```
[full0]
file="/var/log/full.log"
sync=1

[json1]
file="/var/log/firewall_log.json"
sync=1
```

Purpose of This Custom File

- To explicitly control plugin loading instead of relying on defaults.
- To define a clear and auditable path from packet capture (NFLOG) through processing and filtering, to final log output (LOGEMU, JSON).
- To support multiple formats (e.g., text and JSON) for integration with other tools.
- To ensure logs are separated: status logs (**ulogd.log**) vs. packet logs (**iptables.log** or JSON file).

Note:

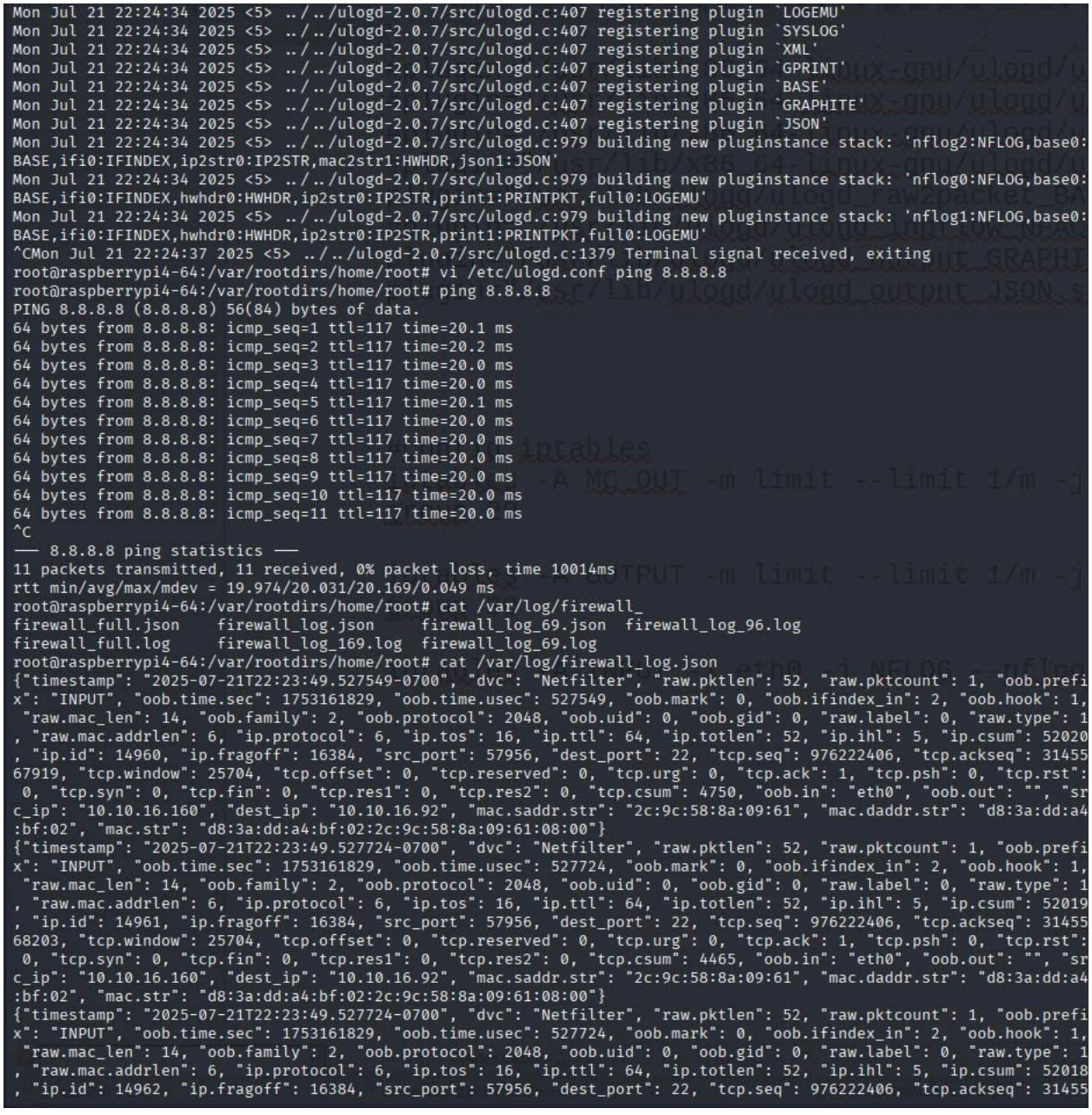
This **.in.user** file is a template. During the build process, it may be renamed to **ulogd.conf** or installed directly into the image at **/etc/ulogd.conf**.

2. JSON LOG OUTPUT CONFIGURATION

After compiling **ulogd** with Jansson support, you can use the **ulogd_output_JSON.so** plugin to write logs in JSON format.

```
[json1]
file="/var/log/full.log"
sync=1
```

The **firewall_log.json** file will contain detailed network traffic logs in structured JSON format, suitable for analytics, dashboards, or SIEM integration



3. LOGEMU Log Output Configuration

3.1 NFLOG:

After compiling **ulogd** with Jansson support, you can use the **ulogd_output_LOGEMU.so** plugin to write logs in LOGEMU format.

```
[full0]
file="/var/log/full.log"
sync=1
```

The **full.log** file will contain detailed network traffic logs in structured LOGEMU format, suitable for analytics, dashboards, or SIEM integration

```
root@raspberrypi4-64:/var/rootdirs/home/root# ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=20.1 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 20.086/20.086/20.086/0.000 ms
root@raspberrypi4-64:/var/rootdirs/home/root# cat /var/log/f
firewall_full.json    firewall_log.json     firewall_log_69.json  firewall_log_96.log   full.log
firewall_full.log     firewall_log_169.log  firewall_log_69.log   flowmgr.err
root@raspberrypi4-64:/var/rootdirs/home/root# cat /var/log/f
firewall_full.json    firewall_log.json     firewall_log_69.json  firewall_log_96.log   full.log
firewall_full.log     firewall_log_169.log  firewall_log_69.log   flowmgr.err
root@raspberrypi4-64:/var/rootdirs/home/root# cat /var/log/full.log
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=136 TOS=10 PREC=0x00 TTL
=64 ID=63967 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232807 ACK=976333182 WINDOW=1542 ACK PSH URGP=0 UID=0 GID=0 MARK=
0
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=136 TOS=10 PREC=0x00 TTL
=64 ID=63967 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232807 ACK=976333182 WINDOW=1542 ACK PSH URGP=0 UID=0 GID=0 MARK=
0
Jul 21 22:46:02 raspberrypi4-64 0x05 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=136 TOS=10 PREC=0x00 TTL
=64 ID=63967 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232807 ACK=976333182 WINDOW=1542 ACK PSH URGP=0 UID=0 GID=0 MARK=
0
Jul 21 22:46:02 raspberrypi4-64 test IN=eth0 OUT= MAC=d8:3a:dd:a4:bf:02:2c:9c:58:8a:09:61:08:00 SRC=10.10.16.160 DST
=10.10.16.92 LEN=52 TOS=10 PREC=0x00 TTL=64 ID=20761 DF PROTO=TCP SPT=57956 DPT=22 SEQ=976333182 ACK=3146232891 WIND
OW=23399 ACK URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 test IN=eth0 OUT= MAC=d8:3a:dd:a4:bf:02:2c:9c:58:8a:09:61:08:00 SRC=10.10.16.160 DST
=10.10.16.92 LEN=96 TOS=10 PREC=0x00 TTL=64 ID=20762 DF PROTO=TCP SPT=57956 DPT=22 SEQ=976333182 ACK=3146232891 WIND
OW=23399 ACK PSH URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=52 TOS=10 PREC=0x00 TTL=
64 ID=63968 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232891 ACK=976333226 WINDOW=1542 ACK URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=52 TOS=10 PREC=0x00 TTL=
64 ID=63968 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232891 ACK=976333226 WINDOW=1542 ACK URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 0x05 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=52 TOS=10 PREC=0x00 TTL=
64 ID=63968 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232891 ACK=976333226 WINDOW=1542 ACK URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=208 TOS=10 PREC=0x00 TTL
=64 ID=63969 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232891 ACK=976333226 WINDOW=1542 ACK PSH URGP=0 UID=0 GID=0 MARK=
0
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=208 TOS=10 PREC=0x00 TTL
=64 ID=63969 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232891 ACK=976333226 WINDOW=1542 ACK PSH URGP=0 UID=0 GID=0 MARK=
0
Jul 21 22:46:02 raspberrypi4-64 0x05 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=208 TOS=10 PREC=0x00 TTL
=64 ID=63969 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146232891 ACK=976333226 WINDOW=1542 ACK PSH URGP=0 UID=0 GID=0 MARK=
0
Jul 21 22:46:02 raspberrypi4-64 test IN=eth0 OUT= MAC=d8:3a:dd:a4:bf:02:2c:9c:58:8a:09:61:08:00 SRC=10.10.16.160 DST
=10.10.16.92 LEN=52 TOS=10 PREC=0x00 TTL=64 ID=20763 DF PROTO=TCP SPT=57956 DPT=22 SEQ=976333226 ACK=3146233047 WIND
OW=23399 ACK URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 test IN=eth0 OUT= MAC=d8:3a:dd:a4:bf:02:2c:9c:58:8a:09:61:08:00 SRC=10.10.16.160 DST
=10.10.16.92 LEN=96 TOS=10 PREC=0x00 TTL=64 ID=20764 DF PROTO=TCP SPT=57956 DPT=22 SEQ=976333226 ACK=3146233047 WIND
OW=23399 ACK PSH URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=52 TOS=10 PREC=0x00 TTL=
64 ID=63970 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146233047 ACK=976333270 WINDOW=1542 ACK URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=52 TOS=10 PREC=0x00 TTL=
64 ID=63970 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146233047 ACK=976333270 WINDOW=1542 ACK URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 0x05 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=52 TOS=10 PREC=0x00 TTL=
64 ID=63970 DF PROTO=TCP SPT=22 DPT=57956 SEQ=3146233047 ACK=976333270 WINDOW=1542 ACK URGP=0 UID=0 GID=0 MARK=0
Jul 21 22:46:02 raspberrypi4-64 0x04 IN= OUT=eth0 MAC= SRC=10.10.16.92 DST=10.10.16.160 LEN=216 TOS=10 PREC=0x00 TTL
```

3.2 NFCT:

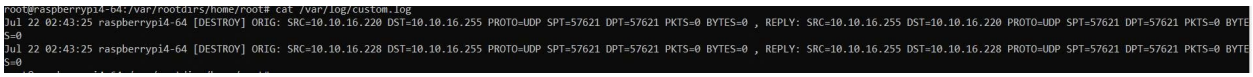
This configuration instructs **ulogd2** to:

- Monitor **connection tracking events** (e.g., TCP connection start/end),
- Convert IP addresses into readable format,
- Format and log the flow information,
- Write it into **/var/log/custom.log.**

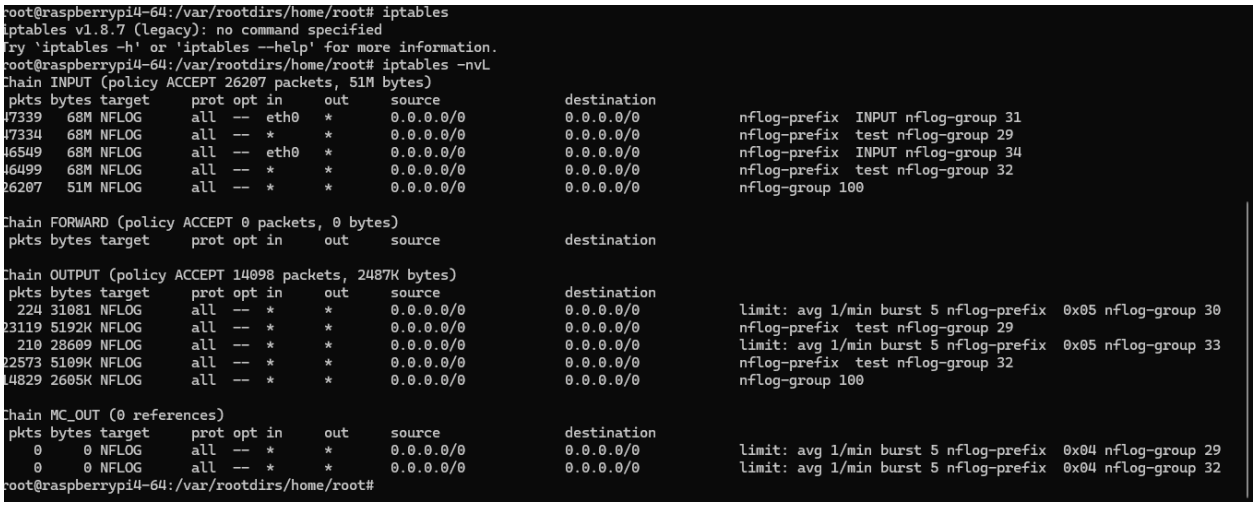
This is especially useful for **auditing**, **network flow analysis**, or **security monitoring**, where connection-level logs are more relevant than individual packets.

```
stack=ct100:NFCT, ip2str1:IP2STR,print1:PRINTFLOW,custom:LOGEMU
[ct100]
[custom]
file="/var/log/custom.log sync=1
```

Below here is an output from ulogd2 using the NFCT (Netfilter Connection Tracking) stack and PRINTFLOW + LOGEMU plugins after configuration:



4. IPTABLES



Divide the traffic into NFLOG groups 29, 30, 31, 32, 33, 34, and 100 to:

- Test multiple groups in parallel.
- Send traffic to different pipelines in ulogd.
- Compare processing speed, log formats, or different outputs.
- Use the limit avg 1/min burst 5 rule to restrict the rate of logged outgoing packets, preventing flooding.
- Supports testing IPv4, IPv6, unicast, and multicast traffic if you combine with appropriate setups.

5. XML LOG OUTPUT CONFIGURATION

5.1 NFLOG:

After compiling **ulogd** with Jansson support, you can use the **ulogd_output_XML.so** plugin to write logs in XML format.

```
stack=log100:NFLOG,base0:BASE,ip2str0:IP2STR,xml1:XML
[log100]
group=100
[xml1]
directory="/var/log/ulogd_xml"
sync=1

~
~
```

The **ulogd-pkt-*.xml** file in folder /var/log/ulogd_xml/ will contain detailed network traffic logs in structured XML format, suitable for analytics, dashboards, or SIEM integration

```
root@raspberrypi4-64:/var/rootdirs/home/root# cat /var/log/ulogd_xml/ulogd-pkt-22072025-024835.xml
<?xml version="1.0" encoding="utf-8"?>
<packet>
<log><when><hour>2</hour><min>48</min><sec>36</sec><wday>3</wday><day>22</day><month>7</month><year>2025</year></when><prefix></prefix><hook>1</hook><hw><proto>0800</proto><src>2c9c588a0961</src></hw><indev>2</i
ndev><payload>450000283178400080069448 0a0a10a00a0a105cd20a00169082a2258c5657ae501004018df60000</payload></log>
<log><when><hour>2</hour><min>48</min><sec>36</sec><wday>3</wday><day>22</day><month>7</month><year>2025</year></when><prefix></prefix><hook>1</hook><hw><proto>0800</proto><src>2c9c588a0961</src></hw><indev>2</i
ndev><payload>450000283179400080069447 0a0a10a00a0a105cd20a00169082a2258c565886501004008d1f0000</payload></log>
<log><when><hour>2</hour><min>48</min><sec>36</sec><wday>3</wday><day>22</day><month>7</month><year>2025</year></when><prefix></prefix><hook>1</hook><hw><proto>0800</proto><src>2c9c588a0961</src></hw><indev>2</i
ndev><payload>45000028317a4000800694460a0a10a00a0a105cd20a00169082a2258c5658aa501004008cfb0000</payload></log>
<log><when><hour>2</hour><min>48</min><sec>36</sec><wday>3</wday><day>22</day><month>7</month><year>2025</year></when><prefix></prefix><hook>1</hook><hw><proto>0800</proto><src>2c9c588a0961</src></hw><indev>2</i
ndev><payload>4500004c317b4000800694210a0a10a00a0a105cd20a00169082a2258c5658aa501804007934000 0dbba962f9c213a626a2933d59e11e7c57f5892d3be9e8a616506b7d87f99a2315539b846</payload></log>
</packet>
```

5.2 NFCT:

```
stack=log100:NFLOG,base0:BASE,ip2str0:IP2STR,xml1:XML

[log100]
group=100

[xml1]
directory="/var/log/ulogd_xml"
sync=1


~
~
```

A. Stack Configuration:

- `log100:NFLOG` → Input plugin using **NFLOG**, group 100.
- `base0:BASE` → Converts raw data into a basic packet structure (Layer 3).
- `ip2str0:IP2STR` → Translates binary IP addresses into human-readable format like **"192.168.x.x".**
- `xml1:XML` → Output plugin that writes the log to **XML** format.

B. Input & Output Plugin Configuration

```
[log100]
group=100
```

- Sets the **NFLOG** group to 100, corresponding to the iptables rule

```
[xml1]
directory="/var/log/ulogd_xml"
sync=1
```

- Logs are written to the directory `/var/log/ulogd_xml`
- `sync=1` → Enables immediate write (instead of buffering)

C. XML Log Output

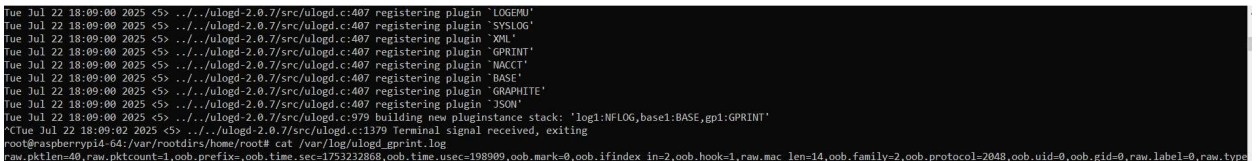
The **`ulogd-pkt-*.xmk`**

6. GPRINT LOG OUTPUT CONFIGURATION

After compiling **ulogd** with Jansson support, you can use the **ulogd_output_GPRINT.so** plugin to write logs in GPRINT format.

```
stack=log1:NFLOG,base1:BASE,gp1:GPRINT
[log1]
group=100

[gp1]
file="/var/log/ulogd_gprint.log"
sync=1
format="%oob.time.sec %ip.saddr -> %ip.daddr %ip.protocol %udp.sport -> %udp.dport\n"
```

The **ulogd_gprint.log** file will contain detailed network traffic logs in structured user defined format, suitable for analytics, dashboards, or SIEM integration



7. LOGROTATE

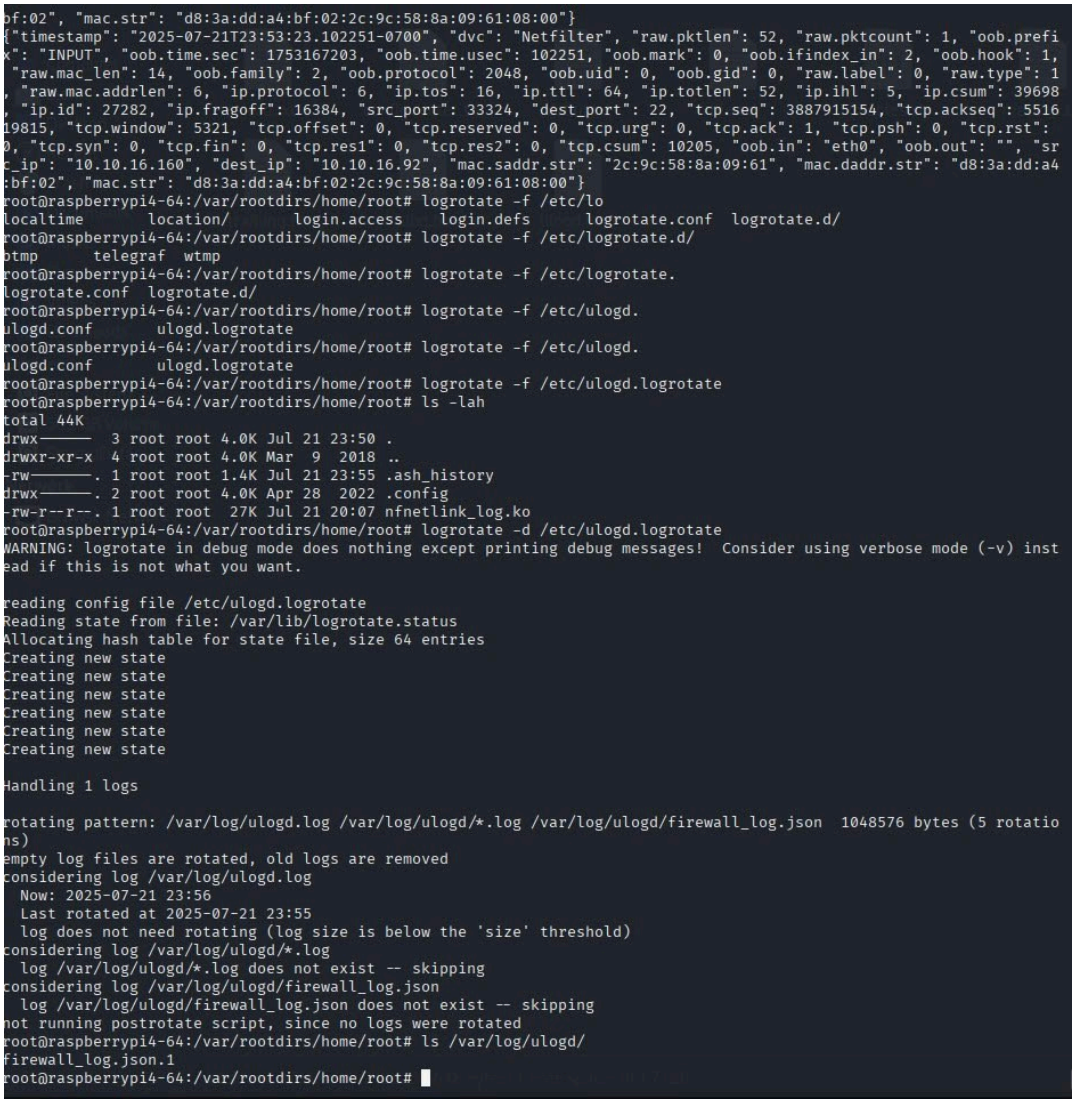
Logrotate is used to manage & rotate the logs based on the age of the file or the file size. It automatically archives old logs, deletes them after a certain number of logs and also creates a

new log file according to the given configuration

Important Logrotate files can be found in the following location's:

- /etc/logrotate.conf : The main logrotate configuration file
- - /etc/logrotate.d : It contains the application-specific configuration files
- ulogd.logrotate.user file:

```
/var/log/ulogd.log /var/log/*.log
/var/log/firewall_log.json {
missingok
size 1M
rotate 5
nodateext
sharedscripts
postrotate
/usr/bin/killall -HUP ulogd 2> /dev/null || true
endscript
}
```


bf:02", "mac.str": "d8:3a:dd:a4:bf:02:2c:9c:58:8a:09:61:08:00"}
{"timestamp": "2025-07-21T23:53:23.102251-0700", "dvc": "Netfilter", "raw.pktlen": 52, "raw.pktcount": 1, "oob.prefi
x": "INPUT", "oob.time.sec": 1753167203, "oob.time.usec": 102251, "oob.mark": 0, "oob.ifindex_in": 2, "oob.hook": 1,
"raw.mac_len": 14, "oob.family": 2, "oob.protocol": 2048, "oob.uid": 0, "oob.gid": 0, "raw.label": 0, "raw.type": 1
, "raw.mac.addrlen": 6, "ip.protocol": 6, "ip.tos": 16, "ip.ttl": 64, "ip.totlen": 52, "ip.ihl": 5, "ip.csum": 39698
, "ip.id": 27282, "ip.fragoff": 16384, "src_port": 33324, "dest_port": 22, "tcp.seq": 3887915154, "tcp.ackseq": 5516
19815, "tcp.window": 5321, "tcp.offset": 0, "tcp.reserved": 0, "tcp.urg": 0, "tcp.ack": 1, "tcp.psh": 0, "tcp.rst":
0, "tcp.syn": 0, "tcp.fin": 0, "tcp.res1": 0, "tcp.res2": 0, "tcp.csum": 10205, "oob.in": "eth0", "oob.out": "", "sr
c_ip": "10.10.16.160", "dest_ip": "10.10.16.92", "mac.saddr.str": "2c:9c:58:8a:09:61", "mac.daddr.str": "d8:3a:dd:a4
:bf:02", "mac.str": "d8:3a:dd:a4:bf:02:2c:9c:58:8a:09:61:08:00"}
root@raspberrypi4-64:/var/rootdirs/home/root# logrotate -f /etc/lo
localtime   location/   login.access   login.defs   logrotate.conf logrotate.d/
root@raspberrypi4-64:/var/rootdirs/home/root# logrotate -f /etc/logrotate.d/
btmp   telegraf wtmp
root@raspberrypi4-64:/var/rootdirs/home/root# logrotate -f /etc/logrotate.
logrotate.conf logrotate.d/
root@raspberrypi4-64:/var/rootdirs/home/root# logrotate -f /etc/ulogd.
ulogd.conf   ulogd.logrotate
root@raspberrypi4-64:/var/rootdirs/home/root# logrotate -f /etc/ulogd.
ulogd.conf   ulogd.logrotate
root@raspberrypi4-64:/var/rootdirs/home/root# logrotate -f /etc/ulogd.logrotate
root@raspberrypi4-64:/var/rootdirs/home/root# ls -lah
total 44K
drwx------ 3 root root 4.0K Jul 21 23:50 .
drwxr-xr-x 4 root root 4.0K Mar 9 2018 ..
-rw-------. 1 root root 1.4K Jul 21 23:55 .ash_history
drwx------. 2 root root 4.0K Apr 28 2022 .config
-rw-r--r--. 1 root root 27K Jul 21 20:07 nfnetlink_log.ko
root@raspberrypi4-64:/var/rootdirs/home/root# logrotate -d /etc/ulogd.logrotate
WARNING: logrotate in debug mode does nothing except printing debug messages! Consider using verbose mode (-v) inst
ead if this is not what you want.

reading config file /etc/ulogd.logrotate
Reading state from file: /var/lib/logrotate.status
Allocating hash table for state file, size 64 entries
Creating new state
Creating new state
Creating new state
Creating new state
Creating new state
Creating new state

Handling 1 logs

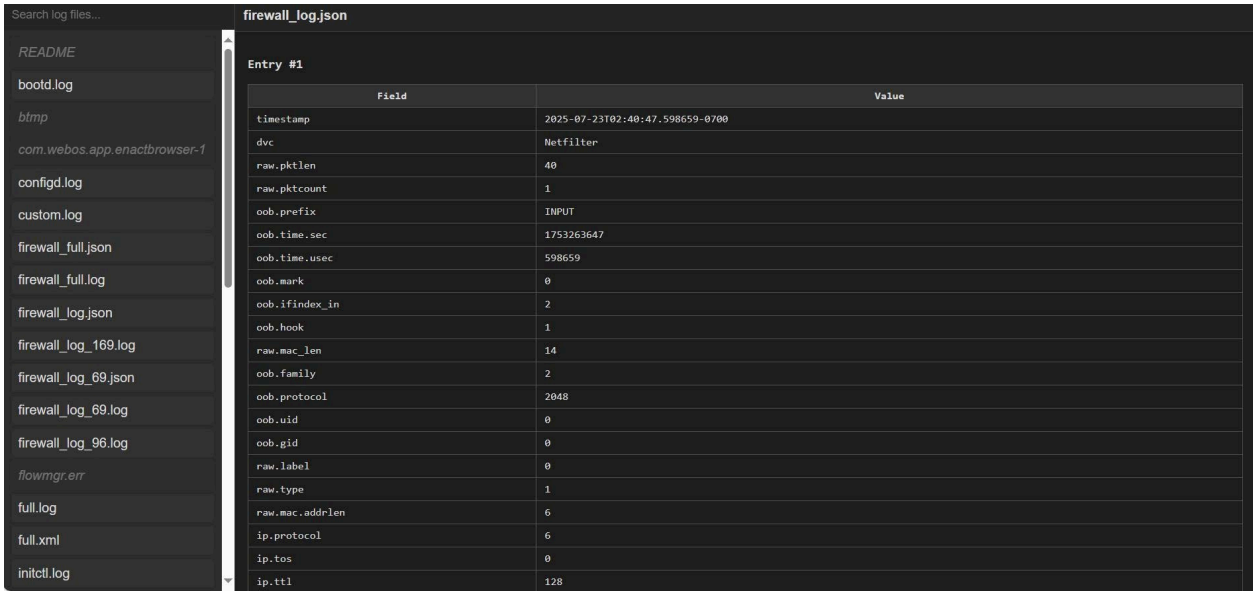
rotating pattern: /var/log/ulogd.log /var/log/ulogd/*.log /var/log/ulogd/firewall_log.json 1048576 bytes (5 rotatio
ns)
empty log files are rotated, old logs are removed
considering log /var/log/ulogd.log
  Now: 2025-07-21 23:56
  Last rotated at 2025-07-21 23:55
  log does not need rotating (log size is below the 'size' threshold)
considering log /var/log/ulogd/*.log
  log /var/log/ulogd/*.log does not exist -- skipping
considering log /var/log/ulogd/firewall_log.json
  log /var/log/ulogd/firewall_log.json does not exist -- skipping
not running postrotate script, since no logs were rotated
root@raspberrypi4-64:/var/rootdirs/home/root# ls /var/log/ulogd/
firewall_log.json.1
root@raspberrypi4-64:/var/rootdirs/home/root#

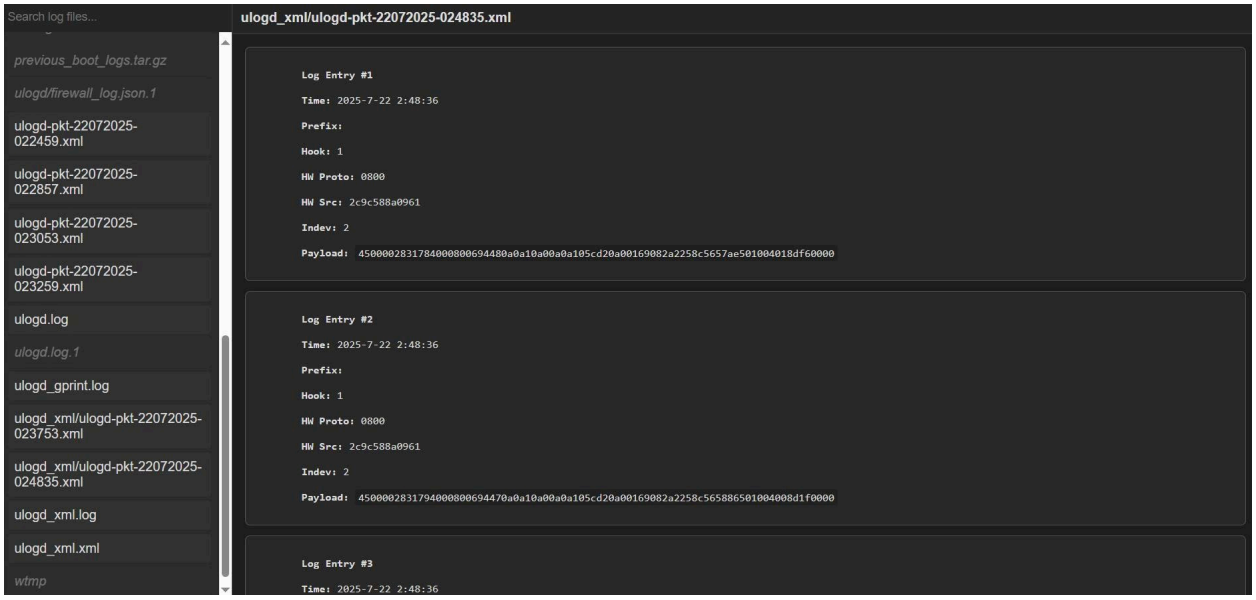
```
root@raspberrypi4-64:/var/rootdirs/home/root# vi /etc/ulogd.logrotate
root@raspberrypi4-64:/var/rootdirs/home/root# logrotate -d /etc/ulogd.logrotate
WARNING: logrotate in debug mode does nothing except printing debug messages!  Consider using verbose mode (-v) inst
ead if this is not what you want.

reading config file /etc/ulogd.logrotate
Reading state from file: /var/lib/logrotate.status
Allocating hash table for state file, size 64 entries
Creating new state
Creating new state
Creating new state
Creating new state
Creating new state
Creating new state

Handling 1 logs

rotating pattern: /var/log/ulogd.log /var/log/ulogd/*.log /var/log/firewall_log.json  1048576 bytes (5 rotations)
empty log files are rotated, old logs are removed
considering log /var/log/ulogd.log
  Now: 2025-07-21 23:57
  Last rotated at 2025-07-21 23:55
  log does not need rotating (log size is below the 'size' threshold)
considering log /var/log/ulogd/*.log
  log /var/log/ulogd/*.log does not exist -- skipping
considering log /var/log/firewall_log.json
Creating new state
  Now: 2025-07-21 23:57
  Last rotated at 2025-07-21 23:00
  log needs rotating
rotating log /var/log/firewall_log.json, log->rotateCount is 5
dateext suffix '-20250721'
glob pattern '-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
renaming /var/log/firewall_log.json.5 to /var/log/firewall_log.json.6 (rotatecount 5, logstart 1, i 5),
renaming /var/log/firewall_log.json.4 to /var/log/firewall_log.json.5 (rotatecount 5, logstart 1, i 4),
renaming /var/log/firewall_log.json.3 to /var/log/firewall_log.json.4 (rotatecount 5, logstart 1, i 3),
renaming /var/log/firewall_log.json.2 to /var/log/firewall_log.json.3 (rotatecount 5, logstart 1, i 2),
renaming /var/log/firewall_log.json.1 to /var/log/firewall_log.json.2 (rotatecount 5, logstart 1, i 1),
renaming /var/log/firewall_log.json.0 to /var/log/firewall_log.json.1 (rotatecount 5, logstart 1, i 0),
log /var/log/firewall_log.json.6 doesn't exist -- won't try to dispose of it
renaming /var/log/firewall_log.json to /var/log/firewall_log.json.1
running postrotate script
running script with args /var/log/ulogd.log /var/log/ulogd/*.log /var/log/firewall_log.json  : "
        /usr/bin/killall -HUP ulogd 2> /dev/null || true
"
root@raspberrypi4-64:/var/rootdirs/home/root#
```

8. VISUALIZATION

This log visualization panel enables users to view, filter, and analyze system and firewall logs easily and intuitively. Logs are displayed in a clear tabular format or separated into individual entries, allowing users to quickly observe key fields such as timestamps, MAC addresses, protocols, hooks, payloads, and packet directions.

Visualizing logs in JSON and XML formats significantly reduces the time required to locate relevant information, helping administrators quickly identify anomalies, policy violations, or track incoming and outgoing data flows. Users can effortlessly browse through multiple log files in the sidebar, compare different entries, and closely monitor packet-level details, making it convenient for firewall inspection, packet debugging, and practical network monitoring.





- File Browsing: Users can browse, select, and open various log files stored in the system for analysis.
- Detailed Entry View: Each log entry can be expanded to show detailed fields, allowing packet-level inspection including protocol types, interface indexes, MAC address length, and payloads.

- Format Support: Supports viewing and parsing of logs in both JSON and XML formats for flexible monitoring and structured analysis.
- Search and Filter: Users can quickly search for specific files or log entries to locate relevant information efficiently.
- Real-time Analysis Support: Helps in monitoring logs generated by firewalls (Netfilter) in near real-time to analyze packet flow, drops, and firewall decisions instantly.

This panel simplifies log management in networking labs, firewall system monitoring, and practical training environments, ensuring that users can analyze and understand system and network behavior with clarity.

V. IMPROVEMENTS:

1. OPTIMIZING LOGGING PERFORMANCE ON LOW-RESOURCE IOT DEVICES

A. Limit Logging Frequency

- Use iptables rate limit: **-m limit --limit 5/min** to prevent log flooding.
- Batch logs with NFLOG: **--nflog-qthreshold 20** aggregates packets before passing to userspace.
- Log only abnormal events: Focus on **NEW, INVALID**, or **DROP** packets, skip established flows.

B. Choose Lightweight Plugins

- Prefer NFCT (flow-based) over NFLOG (packet-based) for lower log volume.
- Log only **NEW** or **DESTROY** events in NFCT.
- Avoid DB outputs; use lightweight formats like JSON or plain text in /tmp.

C. Load Reduction Techniques

- Buffered logging: Use **sync=0** in plugins (e.g. JSON) to delay writes.
- Firewall-level filtering: Only log unwanted or suspicious traffic.
- Write to tmpfs: Store logs in RAM temporarily, then flush to storage periodically.
- Kernel buffer tuning: Increase netlink buffer (e.g. **rmem_max**) to avoid packet loss under load.

D. Example:

Log only 2 events per minute:

```
iptables -A INPUT -m limit --limit 2/min -j NFLOG --nflog-group
5
```

In **ulogd.conf:**

```
stack=log1:NFLOG,...,emu1:LOGEMU
[emu1]
file="/var/log/firewall.log"
sync=0
```

Result: Logs are batched and written less frequently → less I/O, extended flash life.