THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

Syslog over TLS in BusyBox (For Embedded Routers)

Tarun Kundu (Ericsson Software Technology)

About me

- Master's in Informatics and Bachelor's in Electronics
- 21+ years of experience in Software Engineering
- First job: Software engineer at Hughes Software Systems. Worked at Microsoft, AWS, and Cisco
- Embedded systems engineer at Ericsson Software Technology
- I work on embedded router firmware and open-source components for enterprise WAN and private 5G routers.
- This work came from a practical need to secure remote syslog without replacing the lightweight BusyBox logging path.

Executive summary

Need

- Remote router logs must not travel in clear text
- Customers expected RFC 5425-aligned secure logging
- Solution needed to preserve BusyBox's lightweight footprint

Delivered

- Added OpenSSL-backed TLS remote logging to BusyBox `syslogd`
- Added optional public-key pinning for server identity
- Preserved legacy UDP logging and validated end-to-end behavior

Why this matters upstream

- BusyBox `syslogd` is widely used in constrained embedded systems.
- Remote syslog is common, but plain transport exposes operational data.
- A gated TLS path could help embedded deployments secure logs without replacing BusyBox.
- The design preserves existing UDP behavior and keeps TLS optional.

Why secure logging mattered

Existing path

BusyBox `syslogd` was already the standard logging path on the router.

Gap

Remote logs were sent without encryption.

Requirement

Customers expected secure transfer aligned with RFC 5425-style logging.

Decision

Add OpenSSL-backed TLS with optional public-key pinning, while preserving UDP logging.

Secure logging path over TLS

Code Changes

From UDP remote forwarding to optional TLS-backed syslog delivery

Build flags → Runtime path → TLS helper → Public-key Pinning → Validation

Code footprint: focused, modular changes

Small build, runtime, and TLS-helper changes added secure delivery without replacing `syslogd`.

6
files changed

+541
insertions

2
new files

TLS
optional remote path

PIN
optional public-key check

Build

Makefile, Makefile.flags
Config.in
Feature gates
TLS library flags

Runtime

`syslogd.c`
options, TLS state,
retry, send loop

Helper

`sysklogd/tls.c`
TCP connect, TLS
handshake, write, cleanup

API

`sysklogd/tls.h`
Error codes
Helper interface

Build-time integration and feature gates

TLS is compiled only when explicitly enabled; otherwise, the legacy UDP path remains unchanged.

Kconfig

- TLS depends on remote logging
- Pinning depends on TLS

Build Flags

- Add TLS include path
- Link TLS libraries (libssl + libcrypto)
- Export EXTRA_LDFLAGS

Compiled path

- Compile TLS helper only when enabled
- Otherwise: UDP-only remote logging

```
CONFIG_FEATURE_SYSLOGD_TLS=y
CPPFLAGS += -I$(PATH_TO_OPENSSL)/include
EXTRA_LDFLAGS += -L$(PATH_TO_OPENSSL)/lib -lssl -lcrypto
```

User-facing behavior: explicit opt-in modes

UDP

- -R HOST[:PORT]
- Default port: 514
- UDP socket, no TLS state

TLS remote

- -R HOST:6514 –d
- TCP/TLS session
- TLS-backed client path

Pinned mode

- -c <public-key-file>
- Compare peer public key
- Fail closed on mismatch

```
syslogd -R <syslog-ng-host>:6514 -d -c /path/to/pinned-public-key.pem
```

Design intent: TCP/TLS and pinning are explicit opt-in paths; UDP behavior remains unchanged.

Runtime state per remote host

TLS state is tracked independently for each configured remote destination.

remoteHost_t extension

```
typedef struct {
    int remoteFD;
    bool is_tls_enabled;
    SSL_CTX *ctx;
    SSL *ssl;
    unsigned consecutive_failures;
    bool is_pubkey_pin_provided;
    const char *pinned_pubkey_file;
    len_and_sockaddr *remoteAddr;
} remoteHost_t;
```

Session state

`remoteFD`, SSL_CTX, SSL track active TCP/TLS session.

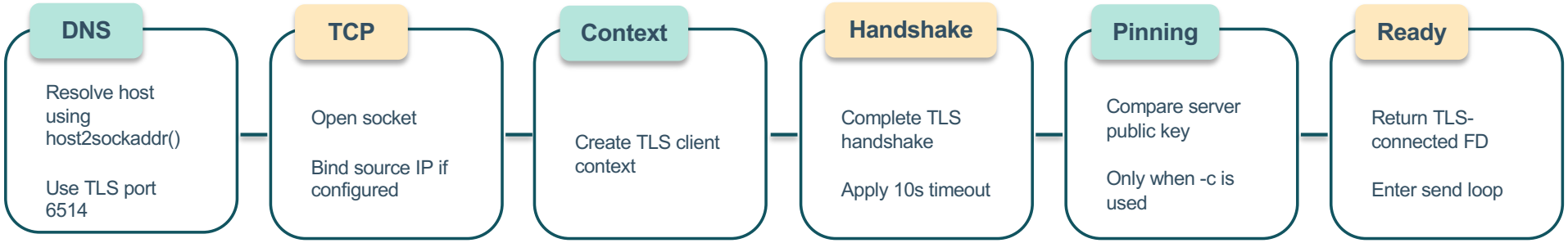
Pinning config

pinned_pubkey_file stores the expected server public-key file when -c is used.

Retry control

consecutive_failures controls backoff and avoids reconnecting on every log line.

TLS connection establishment flow



Failure

Close resources, clear cached address, increment consecutive_failures.

Success

Reset consecutive_failures and reuse the TLS session for forwarding.

New TLS helper module: `sysklogd/tls.c`

open_connection()

- Creates TCP socket
- Supports the existing source-IP bind (-I source-IP option)
- Uses non-blocking connect
- Applies the connect timeout

init_ssl()

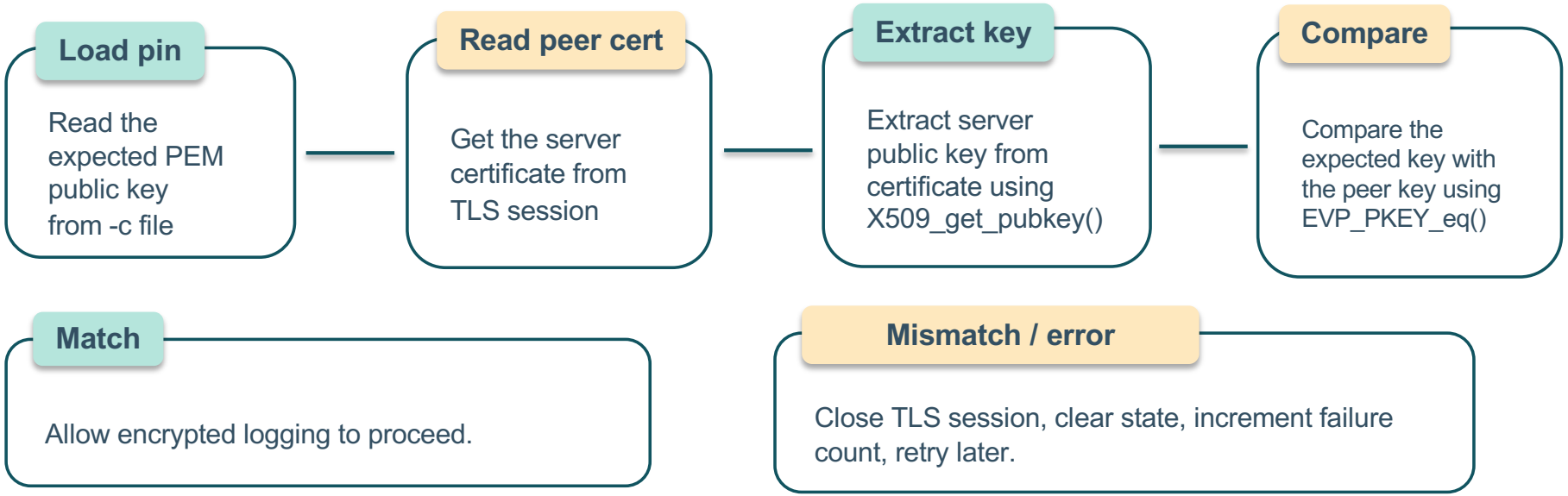
- Creates SSL object
- Attaches socket FD
- Drives `SSL_connect()` with poll
- Enforces 10s handshake timeout

Send + cleanup

- send_TLS_data() wraps `SSL_write()`
- Retries WANT_READ / WANT_WRITE
- Closes TLS state on fatal errors
- Free SSL session and context

Main design consideration: `syslogd.c` keeps logging policy and routing; TLS mechanics stay isolated in tls.c.

Public-key pinning flow



Security note: This is public-key pinning, not full CA-chain, hostname, expiry, or revocation validation.

Message forwarding: dual send path

UDP path

- Open UDP socket if needed
- Forward with sendto(`remoteFD`, sendbuf, send_sz, ...)
- On socket error: close FD, clear address

New TLS path

- Use only after TLS session is ready
- Forward with send_TLS_data(ssl, sendbuf, send_sz)
- On TLS write failure: close TLS state, retry later

Compatibility point: TLS is a parallel branch, not a UDP replacement. TCP/TLS framing must match the receiver mode; remote delivery remains best-effort.

Reconnect and backoff behavior

Problem

- The TLS server may be down, slow, or misconfigured
- Reconnecting on every log line wastes CPU and adds latency

Code behavior

- Track consecutive_failures
- Increase the retry interval using exponential backoff

Embedded-friendly result

- Local logging stays responsive
- Remote delivery retries only at the next deadline and remains best-effort

```
TLS_RETRY_MAX_SEC = 25 * 60
retry interval starts from DNS_WAIT_SEC and doubles after repeated failures
```

Failure handling and cleanup

Failure points

- DNS/address lookup
- TCP connect
- TLS context creation
- TLS handshake timeout
- Pinned-key mismatch
- TLS write failure

Cleanup actions

- Shutdown TLS session
- Free SSL session and context
- Close socket FD
- Set `remoteFD` = -1
- Clear the cached address

Why it matters

- Prevents stale connections
- Enables clean reconnect
- Avoids tight retry loops
- Reduces resource risk

Cleanup order ensures TLS shutdown before socket teardown

Review considerations

Security

- Pinning vs. CA validation
- Dual-pin rotation window
- TLS version + cipher policy

Footprint

- OpenSSL dependency
- Memory/RSS impact
- Smaller TLS stack options

Runtime

- Timeouts and backoff
- Server-down behavior
- Remote message loss semantics

Compliance

- FIPS/compliance gating
- Product crypto policy

Compatibility

UDP logging must work when TLS is compiled in but unused.

Evidence

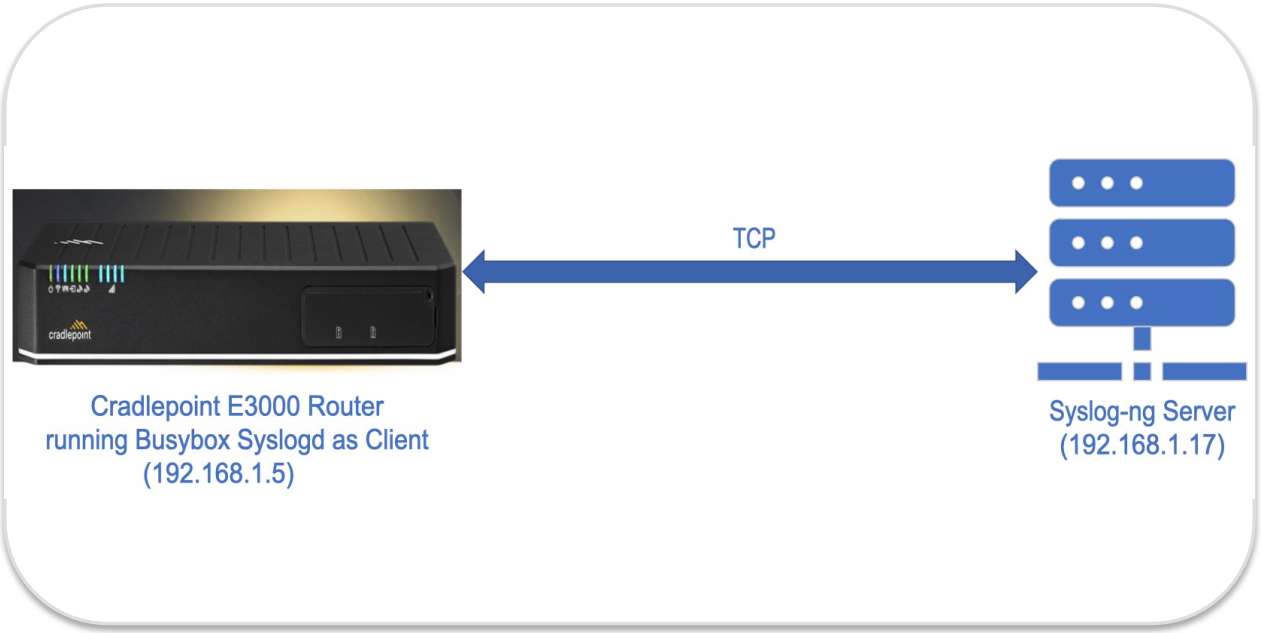
Packet captures, receipt logs, negative pinning tests, server-down tests, and overnight FD/RSS monitoring.

Recorded demo flow

- Start syslog-ng TLS listener on TCP/6514
- Start BusyBox `syslogd` with -R, -d, and optional -c
- Send test log and verify encrypted delivery + server receipt.

TLS secure-flow view

Demo topology



At a glance

- Client: E3000 router
- Server: Syslog-ng host
- Transport: TCP + TLS
- Port: 6514

Syslog-ng server setup

1. Generate a self-signed certificate and private key.
2. Extract the public key used for public-key pinning.
3. Update syslog-ng.conf and attach TLS assets.

```
openssl req -x509 -newkey rsa:2048 ...
openssl x509 -pubkey -noout -in syslog.crt > pubkey.pem
```

Create cert/key and export the pinned public key

```
@version: 3.37
@include "scl.conf"
# Syslog-ng configuration file, compatible with default Debian syslogd
# installation.
# First, set some global options.
source s_net {
        network(
                ip(192.168.1.17) port(6514)
                transport("tls")
                tls(
                        cert-file("/etc/syslog-ng/cert.d/syslog.crt")
                        key-file("/etc/syslog-ng/cert.d/syslog.key")
                        peer-verify(optional-untrusted)
                )
        );
};
destination d_router_log { file("/var/log/router.log"); };
log { source(s_net); destination(d_router_log); };
```

Enable TLS listener

Syslog-ng server startup

Checkpoints

- Start Syslog-ng with the TLS configuration
- Confirm port 6514 is in LISTEN state

```
:) [tarunkundu@fedora syslog-ng]$ ps -ef|grep syslog-ng
tarunku+   84035   82229  0 17:33 pts/3    00:00:00 grep --color=auto syslog-ng
:) [tarunkundu@fedora syslog-ng]$ netstat -na|grep 6514
:( [tarunkundu@fedora syslog-ng]$
:( [tarunkundu@fedora syslog-ng]$ sudo /usr/sbin/syslog-ng -F -v -f /etc/syslog-ng/syslog-ng.conf &
[1] 84038
:) [tarunkundu@fedora syslog-ng]$ netstat -na|grep 6514
tcp        0      0 192.168.1.17:6514       0.0.0.0:*               LISTEN
:) [tarunkundu@fedora syslog-ng]$ ps -ef|grep syslog-ng
root       84038   82229  0 17:33 pts/3    00:00:00 sudo /usr/sbin/syslog-ng -F -v -f /etc/syslog-ng/syslog-ng.conf
root       84039   84038  0 17:33 pts/3    00:00:00 /usr/sbin/syslog-ng -F -v -f /etc/syslog-ng/syslog-ng.conf
tarunku+   84043   82229  0 17:33 pts/3    00:00:00 grep --color=auto syslog-ng
:) [tarunkundu@fedora syslog-ng]$
```

E3000 router readiness

Confirm the E3000 can reach the Syslog-ng host from its WAN interface.

```
[console@E3000-839: /]$ ping 192.168.1.17 -c 3
PING: 192.168.1.17 (192.168.1.17) 56(84) bytes of data.
64 bytes from 192.168.1.17: icmp_req=0 ttl=64 time=0.7 ms
64 bytes from 192.168.1.17: icmp_req=1 ttl=64 time=0.9 ms
64 bytes from 192.168.1.17: icmp_req=2 ttl=64 time=0.8 ms
--- 192.168.1.17 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2033ms
rtt min/avg/max/mdev = 0.700/0.810/0.888/0.188
[console@E3000-839: /]$ wan wan
UID               Type     Prio  Name                    Enabled Port    State        Iface        IP              Other
wan               ethernet 1.0   Ethernet                True    0       connected    wan          192.168.1.5
```

TLS logging (E3000 router to Syslog-ng server)

Flow Start `syslogd` with TLS, send a test log, verify server receipt.

```
[console@E3000-839: /]$ sh
/service_manager # cd /sbin/
/sbin # ls -l syslogd
lrwxrwxrwx    1 root     root            14 Mar 14 17:08 syslogd -> ../bin/busybox
/sbin # ps|grep syslogd
 3048 root      6884 S    grep syslogd
/sbin # /sbin/syslogd -t -l7 -f/var/tmp/syslog.conf -s200 -R 192.168.1.17:6514 -
d
/sbin # ps|grep syslogd
 3084 root      6892 S    /sbin/syslogd -t -l7 -f/var/tmp/syslog.conf -s200 -R
 3094 root      6884 R    grep syslogd
/sbin # exit
[console@E3000-839: /]$ log msg -l INFO "Test log 1"
[console@E3000-839: /]$
```

Router startup + test log

```
Mar 18 18:09:36 192.168.1.5 WAN:73e46871.ConnectorMgr: already reset
Mar 18 18:09:54 192.168.1.5 cpsh: "Test log 1"
Mar 18 18:09:54 192.168.1.5 kernel: [   99.941243] usb 2-1: new SuperSpeed Gen 1 USB device number 3 using xhci-hcd
Mar 18 18:09:56 192.168.1.5 kernel: [  101.972511] usbcore: registered new interface driver cpusb4
```

Server receives the event

Public-key pinning validation

- Pinned startup: Start `syslogd` with -c /path/to/pubkey.pem
- Positive test: Matching server key → TLS session established → log received
- Negative test: Wrong or missing pin → fail closed, clear state, retry later

```
/var/tmp # cat pubkey.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAlTfmA14Rrc31mpbFlLJ8
7mT7xSxlX8j4oTbDxisexdFS2mzeD7nEEKX0wOBOALA5cILXU7ivcOxS3HViwNh/
xH6DQh6sD8vL1TEIy8xDDF3Xivz7k73/0LRnlTHQHpu7/CW2izPHk0B88xTPggMK
XpCFMqNLlqWeKUgTebn2mbk+l1pushmwZe0rEwi7nx2tc1Bv7zrDYdjToAGD2iTQ
yXiN+f4c9twNeHvYVpZizuO3dI8JmdenWOGY7eMcCy/tHy1lEqjCJCblgI5Wl6Qj
qQG3DqHZXqsRVzx3eVZCkSg9CskSvu3NHGdb6I7XuXtGEFdSSZ9/+hYrU8eyYvKM
BwIDAQAB
-----END PUBLIC KEY-----
/var/tmp # /sbin/syslogd -t -l7 -f/var/tmp/syslog.conf -s200 -R 192.168.1.17:651
4 -d -c/var/tmp/pubkey.pem
/var/tmp # ps|grep syslogd
 3236 root      7668 S    /sbin/syslogd -t -l7 -f/var/tmp/syslog.conf -s200 -R
 3282 root      6884 R    grep syslogd
/var/tmp # exit
[console@E3000-839: /]$ log msg -l INFO "Test log 2"
```

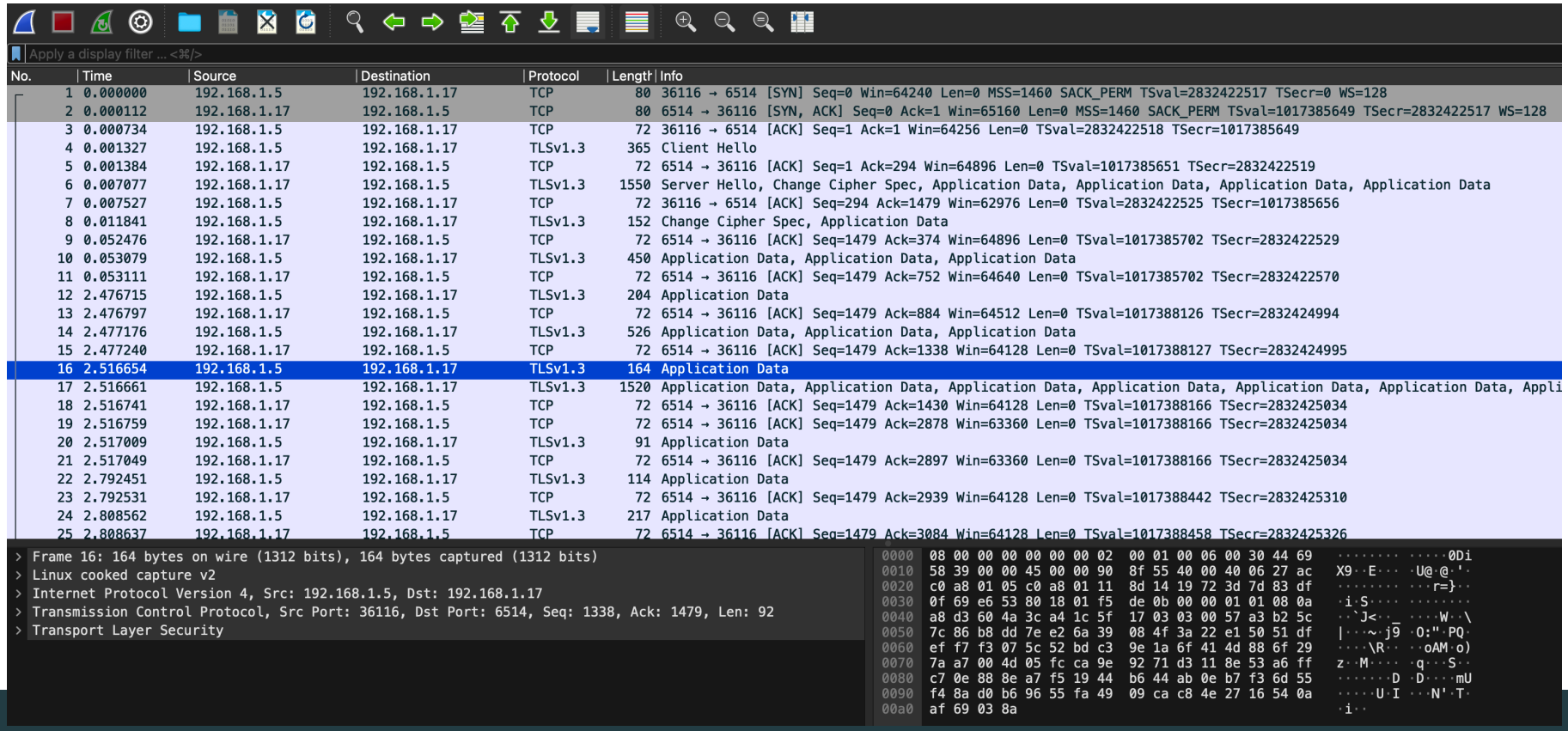
Router startup with pinned key

```
Mar 18 18:35:22 192.168.1.5 cp_stack_mgr: int1: WAN Device reset (5)
Mar 18 18:35:24 192.168.1.5 cpsh: "Test log 2"
Mar 18 18:35:24 192.168.1.5 cp_stack_mgr: int1: Module operating mode set.: Offline
Mar 18 18:35:26 192.168.1.5 kernel: [  138.204236] usb 2-1: USB disconnect, device number 3
```

Pinned log received

Packet capture proof

Capture shows a TCP/6514 connection established, a TLS handshake completed, and subsequent traffic appearing as encrypted application data.



Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.5	192.168.1.17	TCP	80	36116 → 6514 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2832422517 TSecr=0 WS=128
2	0.000112	192.168.1.17	192.168.1.5	TCP	80	6514 → 36116 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1017385649 TSecr=2832422517 WS=128
3	0.000734	192.168.1.5	192.168.1.17	TCP	72	36116 → 6514 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2832422518 TSecr=1017385649
4	0.001327	192.168.1.5	192.168.1.17	TLSv1.3	365	Client Hello
5	0.001384	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1 Ack=294 Win=64896 Len=0 TSval=1017385651 TSecr=2832422519
6	0.007077	192.168.1.17	192.168.1.5	TLSv1.3	1550	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
7	0.007527	192.168.1.5	192.168.1.17	TCP	72	36116 → 6514 [ACK] Seq=294 Ack=1479 Win=62976 Len=0 TSval=2832422525 TSecr=1017385656
8	0.011841	192.168.1.5	192.168.1.17	TLSv1.3	152	Change Cipher Spec, Application Data
9	0.052476	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=374 Win=64896 Len=0 TSval=1017385702 TSecr=2832422529
10	0.053079	192.168.1.5	192.168.1.17	TLSv1.3	450	Application Data, Application Data, Application Data
11	0.053111	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=752 Win=64640 Len=0 TSval=1017385702 TSecr=2832422570
12	2.476715	192.168.1.5	192.168.1.17	TLSv1.3	204	Application Data
13	2.476797	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=884 Win=64512 Len=0 TSval=1017388126 TSecr=2832424994
14	2.477176	192.168.1.5	192.168.1.17	TLSv1.3	526	Application Data, Application Data, Application Data
15	2.477240	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=1338 Win=64128 Len=0 TSval=1017388127 TSecr=2832424995
16	2.516654	192.168.1.5	192.168.1.17	TLSv1.3	164	Application Data
17	2.516661	192.168.1.5	192.168.1.17	TLSv1.3	1520	Application Data, Application Data, Application Data, Application Data, Application Data, Application Data, Appli
18	2.516741	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=1430 Win=64128 Len=0 TSval=1017388166 TSecr=2832425034
19	2.516759	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=2878 Win=63360 Len=0 TSval=1017388166 TSecr=2832425034
20	2.517009	192.168.1.5	192.168.1.17	TLSv1.3	91	Application Data
21	2.517049	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=2897 Win=63360 Len=0 TSval=1017388166 TSecr=2832425034
22	2.792451	192.168.1.5	192.168.1.17	TLSv1.3	114	Application Data
23	2.792531	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=2939 Win=64128 Len=0 TSval=1017388442 TSecr=2832425310
24	2.808562	192.168.1.5	192.168.1.17	TLSv1.3	217	Application Data
25	2.808637	192.168.1.17	192.168.1.5	TCP	72	6514 → 36116 [ACK] Seq=1479 Ack=3084 Win=64128 Len=0 TSval=1017388458 TSecr=2832425326

> Frame 16: 164 bytes on wire (1312 bits), 164 bytes captured (1312 bits)
> Linux cooked capture v2
> Internet Protocol Version 4, Src: 192.168.1.5, Dst: 192.168.1.17
> Transmission Control Protocol, Src Port: 36116, Dst Port: 6514, Seq: 1338, Ack: 1479, Len: 92
> Transport Layer Security

```
0000  08 00 00 00 00 00 00 02  00 01 00 06 00 30 44 69   ........ .....0Di
0010  58 39 00 00 45 00 00 90  8f 55 40 00 40 06 27 ac   X9..E... .U@.@.'.
0020  c0 a8 01 05 c0 a8 01 11  8d 14 19 72 3d 7d 83 df   ........ ...r=}..
0030  0f 69 e6 53 80 18 01 f5  de 0b 00 00 01 01 08 0a   .i.S.... ........
0040  a8 d3 60 4a 3c a4 1c 5f  17 03 03 00 57 a3 b2 5c   ..`J<.._ ....W..\
0050  7c 86 b8 dd 7e e2 6a 39  08 4f 3a 22 e1 50 51 df   |...~.j9 .O:".PQ.
0060  ef f7 f3 07 5c 52 bd c3  9e 1a 6f 41 4d 88 6f 29   ....\R.. ..oAM.o)
0070  7a a7 00 4d 05 fc ca 9e  92 71 d3 11 8e 53 a6 ff   z..M.... .q...S..
0080  c7 0e 88 8e a7 f5 19 44  b6 44 ab 0e b7 f3 6d 55   .......D .D....mU
0090  f4 8a d0 b6 96 55 fa 49  09 ca c8 4e 27 16 54 0a   .....U.I ...N'.T.
00a0  af 69 03 8a                                       .i..
```

Validation matrix

Baseline compatibility

- UDP logging, TLS disabled
- UDP logging, TLS compiled but unused
- Multi-remote regression check

TLS behavior

- TCP/TLS without pinning
- TCP/TLS with a valid public-key pin
- Wrong/missing pin fails closed
- TLS version/cipher mismatch

Reliability

- Server down / DNS failure
- Mid-stream server restart
- Burst + large-message tests
- Overnight FD/RSS monitoring

Takeaways and next steps

What we proved

- Secure remote logging can be added to the existing BusyBox `syslogd` path while preserving legacy UDP behavior, opt-in TLS, and best-effort syslog semantics. This work is a product-validated patch.

Next Steps

Refine it into an upstream-friendly BusyBox contribution with:

- Generic BusyBox Kconfig and build integration
- Clear TCP/TLS syslog framing behavior for RFC 5425 interoperability
- TLS version and cipher policy
- Dependency footprint strategy: OpenSSL now; Mbed TLS, or an in-tree BusyBox TLS approach as future options
- Public-key pin rotation plan
- Maintainer-ready test evidence, including negative tests and long-run FD/RSS monitoring

Codebase: https://gerrit.nordix.org/codeaurora/busybox

Q&A

Codebase: https://gerrit.nordix.org/codeaurora/busybox

Contact: tarun.kundu@est.tech

Tarun Kundu

Embedded Systems Engineer