n8n | community build guide

SELF-HOSTED HOME SECURITY · PLUG-AND-PLAY

Turn your router into a home security operations center

Your network already sees every device, every connection, every threat. This guide gives it a voice, so when something looks wrong at 2 a.m. your phone tells you. Built on free, open-source tools, running on one mini PC.

UniFi | Splunk (SIEM) | n8n | MISP | Telegram alerts | about a weekend

A step-by-step build anyone can follow, with every command, every gotcha, and the AI prompts that do the heavy lifting for you. Based on the architecture by Buddy Rikard.

This is the guide I wish I'd had. It walks you from an empty mini PC to a working pipeline where your router feeds a SIEM, the SIEM feeds an automation engine, and the automation engine messages your phone, with every real-world snag already solved. Where a step is tedious or technical, you get a copy-paste prompt for Claude Code so an AI agent does it for you.

What's inside

Why bother

A security researcher once connected a smart TV to Wi-Fi and watched it phone home to the manufacturer more than 10,000 times in a single day, quietly screenshotting what was on screen for "analytics." The average home now runs 50 to 100 smart devices: bulbs, plugs, cameras, speakers, doorbells. Each one is a tiny computer you don't patch, don't monitor, and can't see inside.

Your router watches all of it. Normally those logs scroll past and vanish. This build captures them, runs real detection logic, and messages your phone the moment something crosses a line, like a new admin login, a firewall block, or a device reaching somewhere it shouldn't. It is the same pattern a corporate security operations center (SOC) uses, shrunk to a mini PC in the corner of your house.

What you'll build

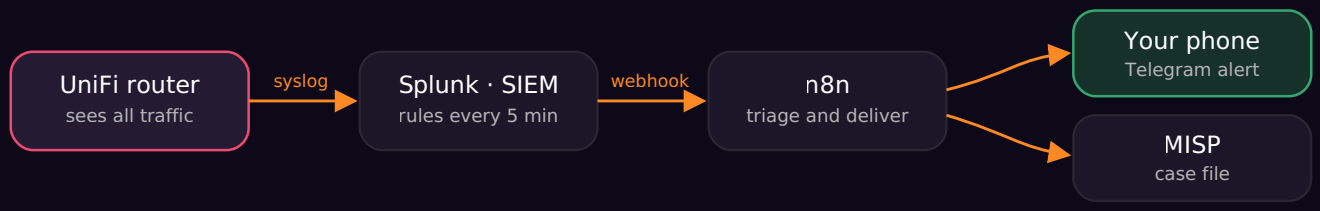
Four open-source pieces, each with one job, wired into a pipeline:

UniFi router, the sensor
Exports everything it sees (device joins, firewall hits, admin changes, threat detections) as standard log events.

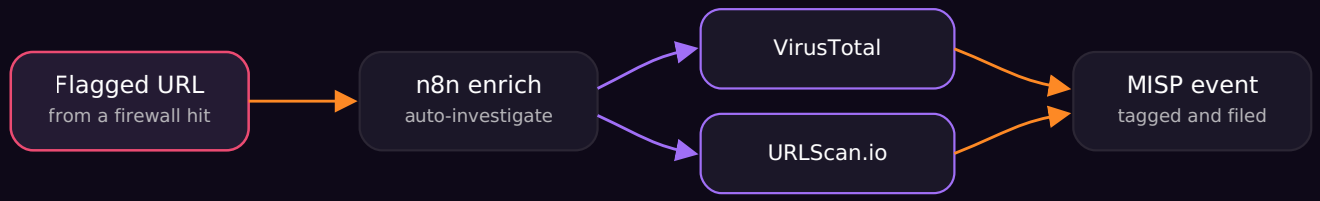
Splunk, the brain (SIEM)
Ingests the logs, runs detection rules on a timer, and decides what is worth escalating.

n8n, the messenger
Receives escalations, makes them human-readable, optionally investigates them, and delivers the alert.

MISP, the case file
A threat-intelligence database that stores a structured record of every incident for later review.



1 — alert pipeline



2 — enrichment pipeline

The whole system: router events flow left to right to your phone, and flagged URLs get auto-investigated and filed.

PLAIN-ENGLISH GLOSSARY

SIEM (security information and event management) is software that collects logs and runs rules to spot trouble, like Splunk. n8n is a visual automation tool you host yourself. MISP (malware information sharing platform) is a tidy database of security incidents. CEF (common event format) is the log format UniFi exports in. A webhook is a URL one app calls to trigger another.

What you need

THING	NOTES
A UniFi gateway	This guide uses a Dream Router 7. Any UniFi gateway with the Activity Logging / SIEM export works. It has to become your main router (phase 0).
An always-on mini PC	A NUC or similar, with 8 GB or more RAM free (Splunk and MISP are hungry) and about 30 GB of disk. Linux is ideal.
Docker, Compose, Git	Installed on the mini PC. Everything runs as containers.
The lab package	Buddy's `self-hosted Cyber Lab and workflows`, which deploys Splunk, MISP, and n8n with one command.
Free API keys	A VirusTotal key and a URLScan.io key (both free tiers), plus a Telegram bot token for alerts.
Recommended	Tailscale on the server (an out-of-band lifeline when you change the network) and Claude Code to run the server steps for you.

THE ROUTER THIS GUIDE USES

Ubiquiti UniFi Dream Router 7 (UDR7)

A desktop 10G all-in-one: router, Wi-Fi, and the Activity Logging / SIEM export this whole build runs on. One box does the routing, the Wi-Fi, and the security logging.

- Wi-Fi 7 (802.11be), tri-band 2.4 / 5 / 6 GHz, up to 5.7 Gbps
- 2.5 GbE ports plus a 10G SFP+ uplink
- WPA3, OSPF, and the UniFi Network features this guide needs

Buy on Amazon

Any UniFi gateway with the Activity Logging / SIEM export works; this is simply the one the guide is built on.

THIS GUIDE ASSUMES A FRESH START

No existing n8n, no existing stack. You run the package's full deploy (`run_all.sh`), which stands up n8n, MISP, and Splunk together on one shared Docker network. It is the simplest path. If you already run n8n, you would deploy Splunk and MISP separately and reuse yours, which is a more advanced variation.

How it flows

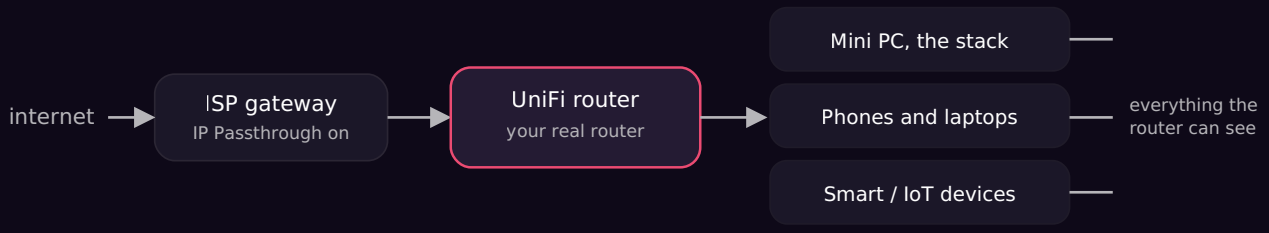
Once it is running, the system works on a quiet 5-minute loop. UniFi streams logs to Splunk continuously. Every 5 minutes Splunk runs your detection rule. If anything notable shows up, it fires a webhook to n8n. n8n triages it, decides severity, and pushes a plain-english alert to your phone, optionally creating a threat-intel record on the way. You do nothing until your phone buzzes.

0

NETWORK · DO THIS FIRST

Put the UniFi router in charge

The router can only protect what it can see, so it has to be the main router for your whole home. On most home internet you keep your ISP's box but stop it from routing, a setting called IP Passthrough, so the UniFi gateway holds the public IP and sees all traffic.



The UniFi gateway sits behind the ISP box but routes for everything, so it observes all device traffic.

Steps

1. Cable it. ISP gateway LAN port to the UniFi gateway's WAN / internet port (a dedicated port, not a regular LAN port). Connect your computer and devices to the UniFi LAN ports.
2. Set up UniFi in the app or console: adopt, update firmware, create your Wi-Fi.
3. Enable IP Passthrough on the ISP gateway. Log into it (often `192.168.1.254` or `192.168.0.1`), find Firewall, then IP Passthrough (wording varies), set the mode to Passthrough / DHCPS-fixed, and select your UniFi gateway from the device list. Save and reboot.
4. Verify. In the UniFi app the gateway's WAN should now show a public IP, not a `192.168.x` address. Run a speed test. You are no longer double-NAT'd.
5. Turn off the ISP gateway's Wi-Fi entirely so only the UniFi radios broadcast.

GOTCHAS, THE NETWORK IS WHERE EVERYONE TRIPS

- WAN versus LAN port. Plugging a computer into the gateway's WAN port (or the ISP feed into a LAN port) means no internet. The WAN port is only for the upstream feed.
- The two-networks, same-SSID trap. If you rename the UniFi Wi-Fi to match your old one for easy device migration, you have to fully disable the old router's Wi-Fi. Otherwise devices silently join the wrong one on a different subnet and nothing works. Confirm the old radio is actually off, since the toggle sometimes does not take the first time.
- Browser red herring. If a router admin page will not load but you are online, you may be on a phone hotspot, or a VPN or proxy browser extension is intercepting local IPs. Test with `ping <router-ip>`: if ping works but the browser does not, it is the browser, so use a different one.

LESSON · KEEP A LIFELINE

Before you change anything on a headless server during this phase, install Tailscale on it. It gives you an out-of-band way back in if a LAN change locks you out, which beats a panicked trip to plug in a monitor.

1

THE SERVER

Stand up the mini PC

1. Put the mini PC on the UniFi network with **a single connection** (wired or Wi-Fi, not both), and give it a **reserved IP** in the UniFi console so its address never changes. Wired is more reliable, and matters more if the server also runs your network DNS.
2. Install **Docker, Docker Compose, and Git**.
3. Confirm you have **8 GB or more RAM free** and plenty of disk.

GOTCHA · DO NOT DUAL-HOME THE SERVER

If the server is plugged in by Ethernet and on Wi-Fi at the same time on the same subnet, both interfaces can grab the same IP and knock it off the network. Pick one link, reserve its IP, done.

ONE COMMAND

2 Deploy the stack

Get Buddy's package onto the server and run the deploy. From the repo root:

```
chmod +x run_all.sh
./run_all.sh
```

This stands up **n8n, MISP, and Splunk** as Docker containers on a shared network called `security-lab` (so they can reach each other by name), and prints the login URLs and credentials at the end. Save those.

LET CLAUDE CODE DO IT

Don't want to SSH around? Paste this into Claude Code, which runs on your computer and can SSH into the server:

```
SSH into my server at <SSH-TARGET> and deploy the self-hosted security lab.
1. Copy the package "<LOCAL-ZIP-PATH>" to the server, unzip it.
2. Pre-flight: confirm Docker, Docker Compose, Git, ~8GB free RAM, and that
   ports 8000/8088/8089 (Splunk), 8443/10080 (MISP), 5678 (n8n) are free.
3. From the repo root run: chmod +x run_all.sh && ./run_all.sh
4. Report the printed URLs and credentials and confirm all containers are healthy.
```

GOTCHAS

- **Resources.** Splunk and MISP together want 8 GB or more RAM. On a tight box the deploy will struggle, so check first.
- **MISP "unhealthy" is usually a false alarm.** The shipped healthcheck pings the wrong port while MISP is actually serving. A one-line compose fix, pointing the healthcheck at the internal port, clears it.

3

AUTOMATION AND ALERTS

Wire up n8n

1. Open n8n at `http://<SERVER-IP>:5678`, create your owner account, and **import the two workflows** from the package: Splunk Security Alert Handler and Firewall Alert, URL IOC Enrichment.
2. **Add credentials** (Credentials, then New): **MISP** (base URL `https://misp-core:443`, "Allow Unauthorized Certs" on), **VirusTotal**, and **URLScan.io**.
3. **Set up alert delivery with Telegram.** Create a bot via @BotFather, add a Telegram node, and set the message to an expression that pulls the alert fields:

```
=🚨 {{ $json.risk_level }} SECURITY ALERT
Event: {{ $json.event_type }}
Host: {{ $json.host }}
Actor: {{ $json.actor }}
Time: {{ $json.received_at }}

{{ $json.recommended_action }}
```

1. **Enable the URLScan "Wait" node** in the enrichment workflow (it ships disabled) so the scan has about 60 seconds to finish, then **activate both workflows**.

GOTCHA · WHY TELEGRAM, NOT GMAIL

Gmail (and any Google login) needs an HTTPS OAuth redirect URL, which a LAN-only server does not have, so the Gmail node will not authorize without a tunnel. A Telegram bot token needs no redirect and just works on a local box. Slack, or SMTP with an app password, are fine alternatives.

LESSON · THE ONE-IP PASSWORD TRAP

Every service here lives on the same server IP with different ports. Password managers key on the IP, so they will happily overwrite your Splunk password with your MISP one. Save each as a separate, clearly named entry (for example, "Splunk admin, server 8000") and choose "create new login," never "replace."

CONNECT THE SENSOR

4 Feed the router's logs to Splunk

ON THE UNIFI SIDE

Go to Settings, then CyberSecure, then Traffic Logging. Set Activity Logging (Syslog) to SIEM Server and enter your server's IP with port 514. Pick the categories that matter (critical, security detections, firewall, triggers, admin activity) and skip the noisy ones.

GOTCHA · THE SIEM SETTING IS NOT WHERE THE DOCS SAY

On current UniFi it is under CyberSecure, then Traffic Logging, not "Control Plane" or "System." Fastest way to find it: type system logs in the Settings search, open the System Logs view, and click "Export to SIEM Server."

ON THE SPLUNK SIDE

Two pieces of plumbing the package does not pre-build: publish the syslog port into the Splunk container, and add the data input. Because Splunk runs as a non-root user, it cannot bind a privileged port below 1024 inside the container, so you map host 514 to container 5514.

LET CLAUDE CODE DO THE SPLUNK PLUMBING

```
On my server, set up Splunk to receive UniFi syslog (CEF over UDP 514).
1. Create a "security" index in Splunk.
2. In splunk/docker-compose.yml add a port mapping: - "514:5514/udp"
   (Splunk runs non-root and cannot bind <1024, so host 514 to container 5514).
   Back up the compose file; recreate ONLY the splunk container.
3. Add a Splunk UDP data input on 5514 to index "security", sourcetype "unifi:cef".
4. Fix timestamps: UniFi sends LOCAL time with no zone, so add a props.conf
   [unifi:cef] stanza pinning _time to the event's UNIFIutcTime field (UTC).
5. Verify events arrive: search  index=security | head  and paste a sample event.
```

TWO GOTCHAS THAT WASTE HOURS

- Zero packets arriving. If a `tcpdump` on UDP 514 shows nothing, it is the network path, not Splunk, usually because the server ended up on the wrong subnet (the same-SSID trap from phase 0). Confirm the server is on the UniFi segment.
- Events land but are invisible. UniFi stamps logs with local time and no timezone, so Splunk dates them hours in the past and they fall outside your search window. The `props.conf` timestamp fix above solves it.

THE DETECTION

5 Create the alert rule

First, see what your router actually sends, in Splunk's search:

```
index=security earliest=-24h
| rex field=_raw "CEF:0\|[^|]*\|[^|]*\|[^|]*\|(?<sig_id>[^|]*)\|(?<event_name>[^|]*)\|(?
| stats count by event_name severity sig_id | sort - count
```

On a quiet home network it is mostly Wi-Fi connect and disconnect noise (severity 1 to 2) plus a few admin and config events (severity 4 to 6). So instead of a rule per event type, use **one broad rule on severity 4 or higher**. It ignores the noise now and automatically catches firewall blocks, threat detections, and critical events the moment they occur.

Save this as a Splunk **alert**: schedule `*/5 * * * *`, time range last 5 minutes, trigger for each result, action Webhook to `http://n8n:5678/webhook/security-alert`:

```
index=security
| rex field=_raw "CEF:0\|[^|]*\|[^|]*\|[^|]*\|(?<sig_id>[^|]*)\|(?<event_name>[^|]*)\|(?
| where severity >= 4
| rex field=_raw "UNIFIclientHostname=(?<unifi_client>\S+)"
| rex field=_raw "msg=(?<unifi_msg>.+)$"
| eval event_type="unifi_event", unifi_event_name=event_name,
       unifi_message=coalesce(unifi_msg,event_name), actor=coalesce(unifi_client,"UniFi"
| table _time event_type severity unifi_event_name unifi_message actor host
```

Finally, teach the n8n handler to understand UniFi events. Add this branch to its Normalize and Triage code node so alerts read cleanly instead of "unknown":

```
} else if (d.event_type === 'unifi_event') {
  const sev = parseInt(result.severity) || 0;
  d.risk_level = sev >= 8 ? 'CRITICAL' : (sev >= 5 ? 'HIGH' : (sev >= 3 ? 'MEDIUM' : 'LO
  d.event_type = result.unifi_event_name || 'UniFi Event';
  d.recommended_action = (result.unifi_message || '') + ' — review in UniFi/Splunk.';
}
```

> WHY "SEVERITY 4 OR HIGHER" IS THE ELEGANT MOVE
>
> One rule, future-proof. It does not need to know every event type, and it auto-covers new threat types the day they first appear. Tune the threshold later if it is too chatty.

PROOF

6 Test it end to end

The alert path. Change a small setting on the router, like renaming a device. That is a Config Modified event. Within about 5 minutes your phone should buzz with a HIGH security alert for Config Modified. That single moment is the whole build paying off: you touched your router and your phone told you.

The enrichment path. The package includes a simulator that sends a fake firewall alert with a URL:

```
SPLUNK_HEC_URL=https://localhost:8088/services/collector/event \
python3 simulate_firewall_alert.py \
  --url "http://testsafebrowsing.appspot.com/s/malware.html" \
  --n8n-webhook "http://<SERVER-IP>:5678/webhook/firewall-url-enrich"
```

Check n8n Executions and MISP Events. You should see VirusTotal and URLScan run and a new MISP event created.

TWO SIMULATOR GOTCHAS

- **Splunk HEC is HTTPS-only.** The script defaults to `http://` and fails with `BadStatusLine`, so set `SPLUNK_HEC_URL=https://…` as shown.
- **VirusTotal 404 on a made-up URL.** VirusTotal's lookup only returns reports for URLs it has already seen, so a fictional test URL returns 404. Use a known URL, like Google's harmless Safe Browsing test page above, and set the VirusTotal node to continue-on-error so real, never-before-seen URLs do not kill the workflow.

MAKE IT REAL

7 Hardening

- **Rotate the default passwords.** The package ships with public defaults, so change the Splunk and MISP admin passwords. It is safe, since Splunk ingestion uses a token and n8n talks to MISP via an API key, not the passwords. Gotcha: change the Splunk password with `docker exec -u splunk …` or it fails on permissions, and update any deploy scripts that embed the old password.
- **Get your own Splunk Developer license** (free, six months, at dev.splunk.com) and drop it in.
- **Make the enrichment resilient** by setting the VirusTotal node to continue-on-error, as above.

- **Optional, network-wide ad and tracker blocking.** If you run AdGuard or Pi-hole on the server, point the router's DHCP DNS at it. Know the trade-off: the server becomes a DNS dependency for the whole network, so keep it on a stable wired link.

Gotchas cheat-sheet

SYMPTOM	REAL CAUSE AND FIX
Computer loses internet when plugged into the gateway	It is in the WAN port, or you are double-NAT'd before passthrough. Use a LAN port, finish IP Passthrough.
Router admin page will not load though you are online	You are on a hotspot, or a VPN or proxy browser extension. Ping the IP: if ping works, it is the browser.
Devices will not auto-reconnect to the renamed Wi-Fi	Security mismatch: new SSID is WPA3 / 6 GHz, old gear is WPA2 / 2.4 GHz. Use a dedicated IoT SSID (WPA2, 2.4 GHz).
Splunk gets zero syslog packets	Network path, not Splunk. The server is on the wrong subnet (the same-SSID trap).
Events in Splunk but not in your search window	UniFi sends local time, no zone. Pin `_time` to the event's UTC field via `props.conf`.
n8n webhook returns HTTP 500	Webhook set to "respond via node" but no Respond node. Switch it to respond immediately.
VirusTotal node errors with 404	URL is not in VirusTotal yet. Set the node to continue-on-error so it proceeds on URLScan data.
Splunk HEC test fails with `BadStatusLine`	HEC is HTTPS-only, so use `https://` in the URL.
Password manager keeps overwriting credentials	All services share one IP. Save each as a distinct named entry, never "replace."
A scary error after a network change	Often a stale symptom that already self-resolved. Run a read-only health check before you tear anything apart.

The Claude Code playbook

The fastest way through the server-side work is to let an AI agent do it. The pattern that worked: **diagnose first, change second, report back**, with clear guardrails like "do not touch

these containers." Reuse the two prompts in phases 2 and 4 above, and for any troubleshooting this framing works well:

REUSABLE TROUBLESHOOTING PROMPT

```
SSH into <SERVER>. <describe the symptom>. DIAGNOSE ONLY, change nothing,
report all output. Check host versus container reachability, routing, the
relevant container logs, and Docker networks. Then recommend a fix but do not
apply it until I approve. Keep a Tailscale lifeline. Do not touch
<production containers>.
```

THE HABIT THAT SAVED THE MOST TIME

Every fix in this build came from a diagnostic, not a guess: ping by IP versus by name (network versus DNS), `docker ps`, reading the actual gateway IP, a MAC-vendor lookup, validating a workflow before trusting it. When something breaks, measure before you change.

Going further, part 2: lock down your IoT

The single biggest upgrade from here is isolating your smart devices on their own network, a dedicated IoT VLAN and SSID. A compromised bulb or camera then physically cannot reach your laptops or phones, and a blocked "IoT to personal device" attempt becomes the highest-signal alert your new pipeline can send. It also fixes the Wi-Fi reconnect pain, since IoT gear gets the 2.4 GHz, WPA2 network it actually wants. That is a guide of its own.

From one builder to another, let's build something brilliant.

Credits and resources

Architecture and the `self-hosted Cyber Lab and workflows` package by Buddy Rikard. Built with open-source n8n, Splunk (free developer license), and MISP, on UniFi hardware. Enrichment via VirusTotal and URLScan.io. Server work accelerated with Claude Code.