

Mavis: A Self-Hosted Cyber News Caller

Amani Dworkin | Home Lab Project | October 2026

Executive Summary

Mavis is a service I built that runs in an isolated Linux container in my home lab. She watches 15 trusted cybersecurity news sources, checks them every two hours during the day, and phones me when something major happens. When the lab's running, she also calls every night at 10pm with a short spoken briefing of the day's top stories, read by a voice generated on the box.

Mavis reads content from the internet all day, and I built her so a compromise wouldn't get an attacker far. Beyond her Tailscale node key she holds no credentials. She can reach only DNS on the lab router, any internet host on port 443, and my phone, and I tested those limits from inside the container. The project is mostly about network segmentation, an encrypted call path from my server to my phone, and using an AI coding agent without trusting it blindly.

Tools: Proxmox, LXC, nftables, Tailscale, Asterisk and SIP, Python, Ollama, Piper, systemd

Key Findings

- The firewall looked right but wasn't doing anything. A higher-level setting was switched off, and I only caught it by testing from inside the container.
- Tightening my network block didn't cut an existing Tailscale connection, and once I blocked that, Tailscale looped traffic through my own public IP. Forcing it through Tailscale's encrypted relays closed both paths.
- Changing the order of fields in the AI model's answer took its accuracy on a small test set from 2 of 6 to 6 of 6, with nothing else changed.

Why I Built It

I'm the co-founder and lead technical author at Obsidia, where we write case studies and develop accompanying tabletop exercises on the world's largest cyber incidents, working alongside leaders in the field. That work made me want to know about big incidents the day they happen. I keep up through social platforms and other sources too, and Mavis is my backstop. She also gave my spare cyber phone a real job: an AI assistant, a bit like the one in Iron Man, that calls me.

What It Does

Mavis checks 15 sources every two hours from 8am to 8pm, including BleepingComputer, The Hacker News, KrebsOnSecurity, threat research from Microsoft, Cisco Talos, and Palo Alto Unit 42, and three CISA feeds covering known exploited vulnerabilities, advisories, and emergency directives. She reads only from that list and never follows links off it.

Each story runs through a filter that looks for named threat groups like ShinyHunters or Volt Typhoon, breaches of major organizations, actively exploited vulnerabilities, and law enforcement action against known groups. Coverage of the same event from several outlets usually gets merged into one item, though not always, which I come back to later.

Call	When	What triggers it
Daytime interrupt	At the next check, every two hours from 8am to 8pm	A tracked threat group hitting a major organization, or a confirmed breach of one, reported by at least three outlets within 24 hours. A law enforcement action against a tracked group, on one source (the law enforcement rule). A new CISA emergency directive. Capped at three calls a day.
Nightly debrief	10pm, when the lab's running	The eight highest ranked stories of the day. If I miss the call, she calls back two minutes later, for up to three attempts in total.

Some big stories go to the debrief on purpose. Exploited vulnerabilities only interrupt me if CISA issues an emergency directive. Unconfirmed claims and stories merged with a weekly roundup column never interrupt, whatever rule they match. The classifier can only demote a story, never promote it. The tradeoff is that I can hear about something up to two hours late, and nothing interrupts me overnight.

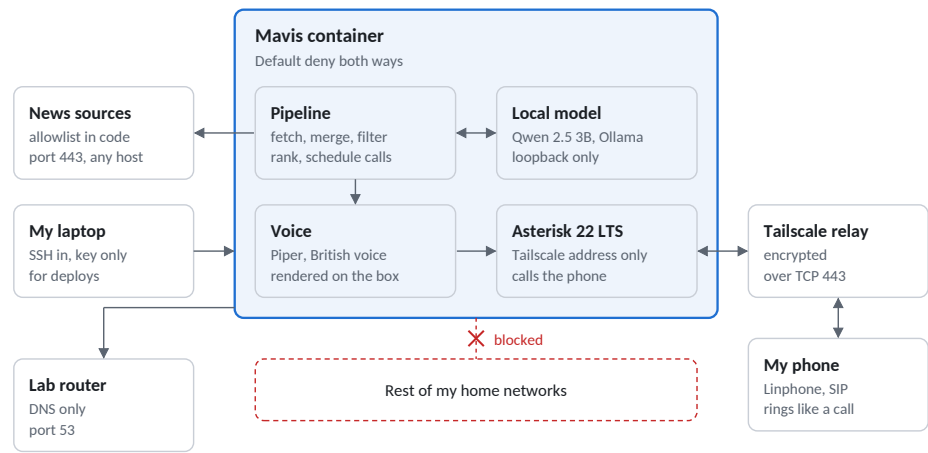
Here's one item from her October 19 debrief:

"...CISA Urges Immediate Ivanti Connect Secure Updates as Honeypots Log 105 Exploit Attempts, from Dark Reading, The Hacker News, and CISA Advisories."

That one came in the debrief, not as a daytime call, because it's an exploited vulnerability with no emergency directive behind it. The top story that night, an FBI investigation into a claimed ShinyHunters breach, stayed in the debrief too. Its coverage got merged with a weekly roundup column, and the classifier flagged it as an unconfirmed claim. Both of those checks override every daytime rule.

The voice comes from Piper, an open source text-to-speech engine, running a British English voice model locally. Mavis went live in October 2026. When the lab's running, she calls every night at 10pm and calls back if I miss it. The lab isn't always on, and on those nights there's no call. Calls ring on the lock screen like a normal phone call, on home Wi-Fi or a hotspot. She hasn't made a daytime call yet. A replay of past events later in this write-up shows that path does fire.

Architecture



Arrows point the way each connection starts. The phone registers with Mavis and Mavis calls the phone, so that path runs both ways. All other traffic, including the rest of my home networks, is dropped.

Mavis is a Python service in one unprivileged Debian LXC container on my Proxmox server. I picked a container over a full VM to save resources on a home server, which means it shares the host's kernel. My lab sits behind its own router inside my home network, so two routers come up in this write-up: the lab router, which also answers Mavis's DNS lookups, and my main home router.

The service fetches the feeds, merges and filters stories, and decides what's worth a call. A small language model, Qwen 2.5 with 3 billion parameters, runs locally through Ollama as a classifier. It listens only on the container's loopback address, and nothing outside the box can reach it. When a call is due, Piper turns the script into audio and Asterisk, an open source phone system, places the call. It reaches my phone through Tailscale, an encrypted mesh VPN, and the phone runs Linphone, a standard SIP calling app.

Security Design

Threat model

Mavis spends her day downloading and processing text written by strangers, and two risks matter most. The first is a malformed feed aimed at the code that parses it. The parser refuses feeds that declare XML entities, caps every response at 5 MB, and parses each source in its own short-lived process with a time limit. A feed that crashes or hangs the parser only costs that one source. Every source still parses as the same user, and a real parser exploit wouldn't be contained the same way. The parser also runs as a dedicated user with no login shell, inside a systemd sandbox, on a box whose only route into my home network is DNS to the lab router.

The second is prompt injection, from a breached source or from a real article quoting an attacker. The classifier's output is limited to fixed fields, mostly yes or no values, and it can only demote a story. An injection could still suppress a real alert by forcing a demotion. Headlines are also read aloud as published, and a breached source could put misleading words into a call. The two breach rules need three separate outlets, but the other two fire on one source: a new CISA emergency directive and the law enforcement rule. A breached news source could abuse the law enforcement rule, and the cap of three calls a day limits how far that goes.

If Mavis were compromised, an attacker would get a box with no stored passwords or API keys, no route into my home network beyond DNS to the lab router, and no path to any device except my phone. Her most valuable secret is her Tailscale node key, and the access policy limits that key to my phone. They'd still get a machine that can reach any host on port 443, which is useful for things like proxying traffic.

Network segmentation

The Proxmox firewall denies all traffic in both directions by default. Only these exceptions are allowed:

Direction	Allowed	Purpose
Outbound	DNS to the lab router only	Name lookups
Outbound	TCP 443 to the internet	News feeds, updates, and the call relay
Inbound	SSH from my laptop only, keys only	Deployments

Two blocking rules sit above the web rule. One drops all traffic to 192.168.0.0/16, which covers every network in my home, with DNS to the lab router as the only exception. The other rejects all outbound IPv6. My first rule set only covered IPv4, and Mavis had a link-local IPv6 address that could still have reached machines on the same bridge. With reject, programs fail instantly and fall back to IPv4 instead of hanging. Both rules log every hit. A hit on either would mean something on Mavis is trying to go where it shouldn't. Proxmox's NDP option is also off for her interface, which stops IPv6 neighbor discovery traffic at the host.

	On	Type	Action	Macro	Interface	Protocol	Source	S.Port	Destination	D.Port	Log level	Comment
0	☒	in	ACCEPT			tcp				22	nolog	SSH from laptop
1	☒	out	ACCEPT			udp				53	nolog	DNS
2	☒	out	ACCEPT			tcp				53	nolog	DNS tcp
3	☒	out	DROP						192.168.0....		info	block LAN
4	☒	out	REJECT						::/0		info	block all IPv6
5	☒	out	ACCEPT			tcp				443	nolog	HTTPS

Mavis's Proxmox firewall rules. Rule 3's destination is 192.168.0.0/16, cut off by the interface. Host addresses redacted.

Proving it works

The first time I tested from inside the container, a request to the lab router got an answer. The container's rules were correct, but the firewall's master switch at the datacenter level was off, and no guest rules were being enforced at all.

After hardening, I ran a verification script inside the container on October 19 and a follow-up check on October 20. Nothing that should be blocked got through:

Test	Expected	Result
TCP to the lab router on ports 22, 80, and 443	Blocked	Blocked
UDP to a closed port on the lab router	Blocked	No response, inconclusive alone
IPv6 link-local ping to all neighbors	Blocked	Blocked
DNS straight to 1.1.1.1	Blocked	Blocked
DNS through the lab router	Answered	Answered
TCP 443 to my own public IP, through the router	No answer	No answer
Proxmox web UI and SSH over Tailscale	Blocked	Blocked
My laptop over Tailscale	Blocked	Blocked
The phone's SIP port over Tailscale	Answered	Answered, SIP 200 OK

Two rows need a note. A router that ignores closed UDP ports looks the same as a firewall drop, and the UDP row doesn't prove much alone. The public IP row only shows that nothing answers there today. No rule blocks it, and a future port forward on 443 would be reachable from Mavis. The firewall log is clearer about the LAN block itself. During the follow-up check it recorded three blocked TCP attempts from Mavis to the lab router, the kernel resending the same connection request:

```
root@pve:~# grep "DST=[lab router] " /var/log/pve-firewall.log | grep -E "PROTO-UDP.*DP
T=9 | PROTO=TCP.*DPT=80 " | tail -n 4
102 6 veth102i0-OUT 20/Oct/2026:23:03:34 -0500 DROP: IN=fwbr102i0 OUT=fwbr102i0 PHYSIN
=veth102i0 PHYSOUT=fwln102i0 MAC= SRC=
DST=[lab router] LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=47023 DF PROTO=TCP SPT=5092
4 DPT=80 SEQ=1293133676 ACK=0 WINDOW=64240 SYN
```

The first of three logged drops of a TCP connection from Mavis to the lab router, from the October 20 follow-up check. Addresses and hardware IDs redacted.

The UDP probe I sent in the same check never showed up in the log. My best guess is Proxmox's log rate limit, since a single UDP packet competes with a lot of inbound noise, but I didn't confirm it, and the UDP row stays inconclusive.

The Tailscale rows were stopped by Mavis's own firewall before Tailscale's policy saw them. Tailscale's built-in policy tests, below, cover the policy layer.

The stateful firewall lesson

At first I allowed Tailscale's direct UDP traffic through the Proxmox firewall, so the phone could connect without a relay. After I widened the LAN block, Mavis still had a working direct connection to my phone over home Wi-Fi. The firewall is stateful and only checks new connections, and Tailscale's keepalive packets kept the old one alive. Clearing the host's connection tracking table would have ended it.

Next I added a rule inside the container to drop Tailscale traffic to my home networks. Tailscale found another path by sending traffic to my home's public IP and letting the router loop it back in. The fix was to remove the Tailscale rule from the Proxmox

firewall entirely, which leaves its default deny blocking all of Tailscale's direct UDP, and to drop that UDP inside the container as a second layer. Tailscale now falls back to its encrypted relay servers over TCP 443.

Closing the Tailscale gap

Tailscale traffic is encrypted inside UDP packets, and the Proxmox firewall can't see inside the tunnel. Tailscale's default policy lets every device reach every other device, which would have given Mavis a path back to my laptop and server.

I replaced the default allow-all policy before Mavis joined. I kept one rule that lets my own devices reach each other, because tagged devices like Mavis are excluded from it. My phone can reach Mavis only on the SIP and audio ports, and Mavis can reach my phone only over UDP. Linphone picks its ports at random and the version on my phone has no setting to pin them, and that grant covers any UDP port. The policy includes tests that run on every save. They confirm Mavis can't reach my server's web interface or SSH, my laptop's SSH or web ports, or the phone over TCP, and Tailscale won't save a version that breaks them.

```
"grants": [
  {"src": ["autogroup:member"], "dst": ["autogroup:member"], "ip": ["*"]},
  {"src": ["phone"], "dst": ["tag:mavis"], "ip": ["udp:5060", "udp:10000-10019"]},
  {"src": ["tag:mavis"], "dst": ["phone"], "ip": ["udp:1-65535"]}
],
"tests": [
  {"src": "tag:mavis", "proto": "udp", "accept": ["phone:5060"]},
  {"src": "tag:mavis", "proto": "tcp", "deny": ["phone:22", "phone:5060"]},
  {"src": "tag:mavis", "deny": ["server:8006", "server:22", "laptop:22", "laptop:443"]}
]
```

Layers inside the container

- nftables inside the container lets only the phone's Tailscale address in, and only on the SIP and audio ports. Outbound over Tailscale, it allows only the phone. It also drops Tailscale's direct UDP traffic, a second layer behind the Proxmox firewall.
- Asterisk 22 LTS was compiled from source after its signature and checksum were verified, and the compiler was removed afterward. The install fails if Asterisk listens on any address other than its Tailscale one.
- Asterisk accepts the phone by its Tailscale IP address. Inside the tunnel, WireGuard guarantees that address can only come from the phone's own key, and it works as authentication without a stored SIP password. The phone registers with Asterisk, which checks every 30 seconds that it's still reachable. SIP and audio are unencrypted inside the encrypted tunnel.

Keeping it patched

Debian security updates install automatically every day over HTTPS, and apt no longer uses plain HTTP at all. Asterisk is built from source and apt can't patch it. It's pinned to a release verified by its signature, and updating it means rebuilding from a newer signed tarball. The Ollama binary and the voice model are pinned by checksum, the AI model is pinned by its sha256 manifest digest, and the Python dependencies, including everything they pull in, are pinned to exact versions. If the model under that name ever changes, the classifier refuses to run. Tailscale comes from its own signed repository and isn't part of the daily automatic updates.

Building and Tuning the Pipeline

The rules decide what's worth a call. The classifier gets exactly one say: it can demote a story from a daytime call to the nightly briefing. A small model fails quietly, which is why it isn't allowed to do more. If its answer is malformed, the rules decide alone.

Before any real call, Mavis ran in dry run mode and wrote every call she would have made to a log. That stage turned up problems no unit test caught:

- **Thread oversubscription.** The model generated 0.34 tokens per second, slow enough to time out. The container could see all six host cores but was only allowed four, and the extra threads stalled waiting on each other. Matching the thread count to the real CPU limit brought it to 9.95 tokens per second, a 29x speedup.

- **Answer field order.** The model fills in its answer fields in the order they're declared. With the yes or no decision first, it committed to an answer before writing out the fields that hold its reasoning, and scored 2 out of 6 on a small labeled test set. Moving the decision to the end raised it to 6 out of 6, and a larger 16 story set later scored 16 of 16. Both sets are small and I labeled them myself. These are early numbers.
- **Story identity collapse.** Twelve unrelated stories shared one identity because each mentioned Microsoft. The first call about any of them would have marked all twelve as already reported. The fix builds identity from names only when they actually pin down a story, like a CVE number or an actor paired with a target, and otherwise from the words of the headline. It showed up in the dry run logs before a single real call was placed.

I also replayed two past events through dry run mode to check the daytime path. CISA's Emergency Directive 24-01 on Ivanti would have triggered a daytime call, as it should. The Salt Typhoon breach of AT&T and Verizon also triggered one, but for the wrong reason. The breach coverage reached two of the three outlets it needs, and the call came from a single article that used the phrase *law enforcement* descriptively and matched the law enforcement rule. The replay and my own debriefs also showed that coverage of one story doesn't always merge. The full October 19 debrief read the same story twice. Both are known gaps in the matching rules.

How I Used AI

I built Mavis with Claude Code, an AI coding agent. I wrote the specification first: the architecture, threat model, source list, watchlist, call script, and a phased build plan. The agent wrote most of the Python and deployment scripts from that spec.

I set rules for how it worked. The coding agent ran on my laptop, never on Mavis, and no account credentials ever touched her. It did use my SSH key to deploy to Mavis and run diagnostics on her, and every command it ran there needed my approval first. It had to present a plan before writing code, stop after every phase for my review, and deploy only through a dry run that I read first. I configured and tested the network design, the firewall rules, and the Tailscale policy myself.

I treated every claim it made as unverified until I saw real output. When I asked it to confirm three earlier fixes, two had never been made and one was half done. A deploy summary named the wrong model, and checking the code showed only the message was stale. Each looked minor, but any one of them could have broken a call without me noticing.

Known Limitations

These are gaps I know about and chose to accept for a home lab.

- The source allowlist is enforced in code. At the network level Mavis can reach any host on port 443, including my own public IP, and a future port forward on 443 would be reachable from her. An egress proxy with a domain allowlist would move that control into the network.
- DNS goes to the lab router, which forwards lookups to the internet, and DNS is a possible exfiltration channel.
- Mavis runs in an LXC container that shares the host's kernel. The host's own patching matters as much as hers, and Tailscale needs the host's tun device passed in. A full VM would isolate her more strongly at a higher resource cost.
- Proxmox's IP filter is off for Mavis, and a compromised container could send traffic from a different address on the bridge.
- The Tailscale grant from Mavis to my phone allows any UDP port, because Linphone picks its ports at random.
- Hits on the LAN block and IPv6 rules are logged, but nothing alerts me. A hit on the LAN block would be the clearest sign of compromise.
- Asterisk, Ollama, and Tailscale aren't covered by the automatic updates, and nothing yet tells me when they need one.
- I SSH in as root with a key. A separate admin account would be cleaner.