Samsung Smart TV — Home Network Pentest Runbook

Enumerate and probe my own Samsung TU700D (Tizen) on my LAN with Kali. Beginner reference — repeatable.

Step 1 — Find the TV on my network

```
sudo nmap -sn 10.0.0.0/24
```

Why/Lesson *Ping sweep lists every host + MAC vendor. I pick the host tagged "Samsung Electronics" (was 10.0.0.234). MAC vendor is how I ID the target — not guesswork.*

Step 2 — Enumerate every open port/service

```
sudo nmap -sV -sC -p- 10.0.0.234
```

Why/Lesson *Full TCP scan + version detection + default scripts. This is the core recon: build a complete inventory of open doors before touching anything.*

Key open ports found: 8001 (remote-control API), 8002 (same over SSL), 7678/8187/9197 (Samsung AllShare UPnP/DLNA + DIAL), 8080 (web server, CORS *), 9080 (Netflix NRDP), 38023 (AirPlay/AirTunes).

Step 3 — Read the results skeptically

Why/Lesson *"closed (reset)" = device actively refuses (LIFX/TV). "filtered (no-response)" = firewall silently drops (the TP-Link). Different security postures.*

Why/Lesson *Service names on filtered/unknown ports (e.g. "teradataordbms") are just nmap's GUESS from the port number, not real detection. Trust them only when state=open and -sV got a reply. The SSL cert (commonName=SmartViewSDK, KR) is the real proof it's the Samsung remote SDK — certs/Server headers don't lie.*

Nmap flags I used

- **-sn** host discovery only (ping sweep), no port scan.
- **-sV** version detection — the service + version string I later use for CVE lookup.
- **-sC** default safe scripts (creds, cert info, misconfigs).
- **-p-** all 65,535 ports (default is top 1,000).
- **-Pn** skip ping, assume up — use on hosts that block pings.

Step 4 — The control API + its auth model

```
pip install samsungtvws --break-system-packages
python3 -c "from samsungtvws import SamsungTVWS; SamsungTVWS(host='10.0.0.234', port=8002, token_file='tok.txt',
name='KaliRemote').send_key('KEY_VOLUP')"
```

Why/Lesson *Using send_key over the official protocol = using a FEATURE, not exploiting a bug. The security question is only: does it require authorization? On 2016+ Tizen it pops an on-screen "Allow?" prompt + issues a token. That prompt = good security.*

Step 5 — Check the auth posture WITHOUT lighting up the screen

```
curl -s http://10.0.0.234:8001/api/v2/ | python3 -m json.tool
```

Why/Lesson *The prompt only fires on the CONTROL channel. This read-only device-info endpoint is unauthenticated and silent. Field TokenAuthSupport:true = auth enforced (not weak). Also returns model/firmware for CVE lookup.*

Concept — how I knew to hit http://IP:8001/api/v2/

URL anatomy: http:// = protocol (nmap showed HTTP replies) | 10.0.0.234 = host (talk to IP directly, no DNS) | :8001 = port (which service; must specify non-default ports) | /api/v2/ = path (which resource).

Finding paths: nmap gives the port; the PATH I get by (1) deduction — apps control TVs via an API, so an API endpoint must exist; (2) convention — APIs live at /api/ and are versioned /v1//v2/; (3) brute force — gobuster/ffuf spray a wordlist of common paths. curl fetches ONE path I chose; it doesn't discover on its own. Which version (v2 not v1) came from docs/source + testing 200-vs-404, not deduction.

Step 6 — CVE lookup route (my exact model)

Model UN75TU700D / 21_KANTSU2E (2021 Tizen), remote API 2.0.25, developerMode=0, TokenAuthSupport=on. Inputs for any search = vendor + product + version.

Why/Lesson *KEY LESSON: searchsploit came up EMPTY for this TV (its one hit, "SmartViewer BackupToAvi," is unrelated Windows CCTV software = false positive). searchsploit only indexes Exploit-DB = published exploit CODE, which barely covers consumer IoT. Empty searchsploit does NOT mean "no vulns." For TVs the info lives in web searches, blogs, NVD, and papers — that's where I actually found everything below.*

Where I actually look (in order): 1) Web search: "samsung tizen smart tv CVE / vulnerability / exploit" — most productive; surfaces researcher blogs + news. 2) NVD (nvd.nist.gov) + OpenCVE / CVE Details — formal CVE records, version ranges, severity. 3) Researcher blogs (Bishop Fox, etc.) + seclists Full Disclosure + arXiv. 4) searchsploit / Metasploit LAST — check but expect empty for IoT (they shine for server software, not TVs).

```
searchsploit samsung tizen      # check it, but expect No Results for a TV
```

Why/Lesson *Then judge each hit against MY config, don't just collect CVEs: (1) Bishop Fox SDB command-injection RCE fits my OS BUT needs developer mode ON + attack from the dev-host IP — my dev mode is OFF, so it doesn't apply. (2) Wi-Fi Direct MAC-only auth bypass = 2015 models, not my 2021. (3) Older Tizen zero-days = patched. A real CVE can exist for my OS yet not apply because preconditions aren't met.*

Firmware showed "Unknown" over network → get exact version from TV: Settings > Support > About This TV, then match to CVE lists.

Verdict & fix

Verdict: Patched, dev-mode-off, token-auth-on 2021 TV is NOT trivially owned by public CVEs. The one current RCE is gated by developer mode, which is off = my protection. "Properly requires consent" is a valid finding.

Remediation: Keep developer mode OFF; keep firmware updated; put the TV on a separate IoT/guest VLAN so its many services (UPnP, AirPlay, NRDP) are isolated from PCs/phones.

Repeat checklist

1. nmap -sn to find the Samsung IP. 2. nmap -sV -sC -p- <ip> to enumerate. 3. curl http://<ip>:8001/api/v2/ to read model + TokenAuthSupport (silent). 4. Research CVEs: web search + NVD first, searchsploit last (expect empty for IoT), using model + firmware. 5. Match CVE preconditions to my actual config before concluding. Same workflow fits any device — swap the IP and research its services.