

public-research/cecbot/report.md at main · deepfield/public-research

github.com/deepfield/public-research/blob/main/cecbot/report.md



CECbot: a TV box botnet that grabs the remote and maps the house

Nokia Deepfield Emergency Response Team (ERT)

First published: 2026-03-21

Executive summary. CECbot is a previously undocumented DDoS botnet targeting Android TV boxes. It is the operational successor to [Katana](#), sharing infrastructure but no code. Built as a native Android application rather than a Mirai ELF binary, it uses Curve25519 + Ed25519 + ChaCha20-Poly1305 for C2 encryption, implements 9 persistence layers, and carries 11 DDoS attack methods with HTTP/2 and dynamic TLS support. Two capabilities stand out. First, CECbot is, to our knowledge, the first documented malware to use HDMI Consumer Electronics Control (CEC), giving the operator full control of the HDMI bus including the ability to put the connected TV to sleep. Second, it can map the victim's home network via automated subnet scanning and ARP correlation, turning a compromised TV box into a reconnaissance platform on the local network. The bot is delivered through the same residential-proxy-to-ADB chain [documented by Synthient](#) for Kimwolf, which places it on a trusted subnet where the network scanning is most

useful. CECbot is the latest evidence that the proxyware SDKs bundled with uncertified AOSP devices remain an ongoing threat: each generation of malware exploiting this delivery chain is more capable than the last. These devices are not confined to homes — they are anywhere someone wanted a cheap screen and didn't ask questions about the box behind it. The LAN discovery is a curiosity in a living room. On a hospital or corporate network, it is something else entirely.

i Note

This report is based on static analysis of the CECbot APK and associated infrastructure. We intend to revisit it as we collect additional observations on botnet population size and attack activity.

In mid-March 2026, the operator behind [Katana](#) — a Mirai fork we have been tracking since February, with at least 30,000 bots and 150 Gbps attack volumes — deployed something entirely new alongside it. The new APK used the same package name (`com.google.android.update`) and targeted the same devices, but did not share a single line of code with Katana.

Katana is a C binary compiled with GCC 4.2.1 for uClibc, with single-byte XOR encryption and a plaintext C2 protocol. What replaced it is a clean-sheet Android application — a Java C2 layer, a native JNI attack engine, Curve25519 key exchange, Ed25519 server authentication, and ChaCha20-Poly1305 traffic encryption. The cryptographic primitives are the same ones used by Signal and WireGuard. Cryptographically, the gap between Katana and its replacement spans several decades of protocol design. Katana brought a Caesar cipher to a cryptography fight. CECbot brought Noise.

We named it after one of its two most distinctive features. The first explains itself once you know what CEC stands for. The second is what the operator can see while the user cannot. But first: what is it, what does it do, and how does it stay.

Sample

field	value
SHA-256	b3c1d5fc273d19556b09f935b9b09b782b113b98a8a010ebcbb5de5bfce77e67
type	Android APK
size	165 KB (344 KB uncompressed)
package	com.google.android.update
build date	2026-03-20 23:16 UTC
architectures	armeabi-v7a, arm64-v8a

field	value
NDK	r25c (Clang/LLD)
obfuscation	R8 minification

The APK contains 14 files: a single DEX (93 KB), two architecture variants each of the attack engine ([libattack.so](#), ~111 KB) and watchdog ([libwatchdog.so](#), 5-7 KB), minimal resources, and signing metadata. At 165 KB compressed, the entire botnet is smaller than most app launcher icons. It would fit in a single iMessage attachment with room left for a "u up?" — which, given the CEC standby command, is not entirely off-brand.

No embedded Tor binary ([libtor.so](#)) is present in this build. The Tor fallback code is complete but the binary it invokes is absent: an emergency exit with the door installed but no handle. It is likely delivered via a post-installation update (command `0x11`) or reserved for future builds.

From Katana to CECbot

All observed delivery targets ADB-exposed Android TV devices via the same attack surface [documented by Synthient](#) for Kimwolf: many uncertified Android TV boxes ship with proxyware SDKs preinstalled, which both expose the device's own ADB port to proxy clients and allow tunneling into the local network to reach other devices with unauthenticated ADB. The same proxy tunnel that delivers the bot places it on the local network, on a trusted subnet. The bot has no self-propagation capability; initial distribution is handled by external ADB exploitation scripts. Once installed, it can scan the local network (see), but does not spread autonomously.

The evidence linking CECbot to the Katana operator:

indicator	Katana	CECbot
package name	com.system.update → com.google.android.update	com.google.android.update
.st TLD	thespacemachines[.]st	sdkconnect121[.]st
shared C2 IP	91.92.241[.]12:6969	91.92.241[.]12:10213

Same package name, same TLD preference, same target population, shared C2 server. The attribution is circumstantial in the way that gravity is theoretical.

[91.92.241\[.\]12](#) hosts both the Katana C2 (port 6969, still receiving attack commands as of 2026-03-21) and the CECbot C2 (port 10213) — different botnets, same server, different ports. The operator is running both in parallel during the transition.

The Jackskid operator's `komaru` and `komugi` ADB stagers explicitly uninstall `com.google.android.update`, confirming the two operations compete for the same devices. Being explicitly targeted for removal by a rival botnet is, in its way, a form of industry recognition.

But the interesting part is not what stayed the same. It is what changed.

The architectural leap

Most Android DDoS bots, including the dozens of Mirai forks we track, run cross-compiled ELF binaries dropped to `/data/local/tmp`. They are Linux programs that happen to run on Android. Some, like the Jackskid and MossadProxy families we [documented previously](#), have evolved to wrap the ELF payload in a companion APK for delivery and mutual persistence — but the APK is a thin dropper, not an Android application. The native binary it exec's still has no awareness of the Android framework: no access to AlarmManager, JobScheduler, foreground services, broadcast receivers, or the vendor-specific power management interfaces that govern whether an app is allowed to run in the background.

CECbot is built as a native Android application. The Java C2 layer handles persistence, scheduling, device profiling, and command dispatch. The native attack engine is loaded via JNI, not exec'd from `/tmp`. The watchdog is a PIE executable disguised as a shared library. The APK registers broadcast receivers, schedules jobs, acquires WakeLocks, and manipulates OEM-specific settings. None of this is available to a raw ELF binary.

This matters because Android TV boxes are not Linux routers. They have battery optimization, phantom process killers, background execution limits, and per-vendor power management policies. Katana fought the platform. CECbot uses it.

C2 infrastructure

type	value	notes
domain	<code>sdkconnector[.]com</code>	primary C2; fast-flux (18 IPs)
domain	<code>sdkconnect121[.]st</code>	secondary C2
port	10213/TCP	hardcoded
Tor	<code>c2kxpjr7cux7fqrfrmimsz7rtq527xauw627xrjojimt66nwxqvrqbuyd[.]onion</code>	fallback, port 80

`sdkconnector[.]com` resolves to 18 IPs via DNS round-robin, spread across multiple hosting providers (OVH, IBM Cloud, CloudBackbone, Baxet Group, and others). The full IP list is in [iocs/ips.csv](#).

The C2 can push new domains at runtime (`0x30`), persisted in Android SharedPreferences (`d_prefs`), XOR-encoded and base64-wrapped. The operator can rotate infrastructure without redeploying the APK.

C2 protocol

Every message uses an 8-byte header:

```
[BE EF] [version: u8] [flags: u8] [payload_length: BE u32]
```



The frame magic is `0xBEEF`, present in every frame, handshake and data alike. Flags encode deflate compression (bit 0) and ChaCha20-Poly1305 encryption (bit 1). Maximum payload: 1 MB.

Handshake

The handshake uses Curve25519 for key exchange and Ed25519 for server authentication. The bot will not accept a C2 server that cannot prove possession of the operator's private signing key — a level of cryptographic discipline rarely seen in DDoS botnets, and entirely absent from the Katana it replaced.

```
Bot → C2:  BE EF 00 [ephemeral_pubkey: 32B] [SHA-256(server_static_pubkey): 32B]
C2 → Bot:  BE EF 00 [ephemeral_pubkey: 32B] [Ed25519_signature: 64B]
```



Session keys are derived via HKDF-SHA256 with four labels (`c2b`, `b2c`, `nc2b`, `nb2c`), producing independent encryption keys and nonces for each direction. Each session uses a fresh ephemeral keypair, providing forward secrecy. Katana's C2 protocol is plaintext TCP with a 4-byte magic (`0x00000001`) and no encryption at all.

Tor fallback

The code implements a Tor-based fallback, triggered after 2+ consecutive connection failures. The current build does not bundle `libtor.so` (see), so this path is inactive, but the logic is complete and will activate if the binary is delivered via a later update. The sequence:

1. Launches `libtor.so` with `--SocksPort 9050`
2. Waits up to 90 seconds for `Bootstrapped 100%`
3. SOCKS5 connects to the `.onion` address on port 80

4. Sends `GET /c HTTP/1.0`
5. Decrypts the response with ChaCha20-Poly1305 (static key)
6. Validates timestamp (rejects responses older than 7 days or more than 1 hour in the future)
7. Extracts a signed domain list and feeds it into the C2 resolver

Even if all clearnet domains are seized or sinkholed, the operator can push new C2 addresses through the hidden service. Katana's fallback is a hardcoded IP address.

What it does

DDoS

The native attack engine ([libattack.so](#)) registers 11 attack handlers, loaded via JNI from the Java C2 layer. Katana also supports 11 methods, but the implementations are entirely different: Katana uses Mirai-derived C code compiled with GCC 4.2.1; CECbot uses NDK r25c with Clang/LLD and dynamically loads TLS from the Android system libraries.

idx	type	description
0	UDP game flood	8 sub-types: Source Engine, Quake3, SAMP, TeamSpeak3 INIT, Discord voice, random, amplification
1	UDP random flood	Random payload (8-65,000 bytes), pure volumetric
2	UDP custom flood	Operator-specified payload, or 512-byte random fallback
3	UDP large-packet	Random payload 900-1,397 bytes, MTU-sized
4	TCP connection flood	Non-blocking multi-connect (default 500 concurrent), slowloris-style
5	Raw TCP SYN flood	SOCK_RAW, crafted IP/TCP headers, spoofed sources
6	Raw TCP+TLS flood	SOCK_RAW, 80% random / 20% simulated TLS ClientHello
7	TCP protocol flood	9 protocols: SSH, FTP, SMTP, IMAP, POP3, MySQL, IRC, FiveM, LDAP
8	Raw TCP crafted	SOCK_RAW + stream hybrid, <code>xxxxxxxx</code> payload pattern
9	IP protocol flood	SOCK_RAW(0xFF): ICMP echo, GRE, ESP/IPSec, VRRP, L2TP
10	HTTP/HTTPS L7 flood	HTTP/1.1 + HTTP/2, dynamic TLS, 660+ referers, 8 browser families

The most significant upgrade over Katana is attack 10: a full HTTP/HTTPS L7 flood that dynamically loads TLS from `/system/lib{64}/libssl.so` and `/system/lib{64}/libcrypto.so`. Katana's HTTP flood sends plaintext HTTP/1.1 into HTTPS listeners — which is, at best, a polite knock on the wrong door. CECbot speaks TLS, HTTP/2 (`PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n`), generates browser-realistic `Sec-CH-UA` headers across 8 fingerprint families (Chrome, Firefox, Edge, Safari, Brave), and includes a pool of 660+ legitimate referer URLs spanning major websites from `google.com` to `virustotal.com` — citing the service most likely to flag you as malware in your own HTTP headers is a choice.

The protocol flood (attack 7) shares capabilities with Katana but not code: SSH banner spoofing with `PuTTY_Release_0.79` through `0.83`, SMTP relay against the same placeholder hostnames (`mail.server.local`, `smtp.domain.net`), and the same `CitizenFX/1` FiveM API calls. Whether the CECbot author had access to Katana's source, or whether both drew from the same reference material, the attack templates are functionally identical despite the code being entirely different.

Proxy

Beyond DDoS, CECbot is a residential proxy platform. The proxy infrastructure is decoupled from the C2 — the operator specifies a separate proxy backend via the command payload.

SOCKS5 proxy (`0x14–0x16`): Binds a `ServerSocket` on the operator-specified port with a 50-thread pool. Standard SOCKS5 with optional RFC 1929 username/password authentication. Traffic flows directly from the SOCKS client through the bot to the destination without passing through the C2. Only useful if the bot has a reachable IP (public, or same LAN as the client).

Reverse proxy (`0x17–0x18`): The bot connects outbound to operator-specified proxy infrastructure (not the C2), making it work behind NAT. On connect, it sends a 38-byte registration containing a 32-byte per-bot auth key, the bot's external IP (resolved via `ipify/checkip`), and a 2-byte marker. The external IP enables the proxy backend to geo-route traffic through specific residential IPs. The backend then multiplexes TCP tunnels through the bot using a binary framing protocol with session IDs. The bot also reports proxy status back to the C2 via the `0xBEEF` channel (command `0x87`), giving the operator visibility across both the C2 and proxy connections.

The per-bot auth key, external IP registration, and geo-routing capability indicate this is designed for a multi-bot proxy service. The architecture is identical to commercial residential proxy SDKs: infected devices become exit nodes, the proxy backend handles routing and customer multiplexing, and the C2 manages the bot fleet separately. The operator who built CECbot to exploit the proxyware-to-ADB delivery chain is, in turn, running the same business model on the devices it compromises.

How it stays

Persistence (9 layers)

CECbot implements 9 persistence mechanisms. Where Katana relies on Magisk modules and system partition copies, CECbot uses the platform's own scheduling and service infrastructure against it.

layer	mechanism	detail
1	Foreground service + WakeLock	Blank notification (channel "s", content " " — minimalism Marie Kondo would respect); 24-hour partial WakeLock
2	AlarmManager	Two alarms at 15-minute intervals fire <code>BootReceiver</code> with WATCHDOG action
3	JobScheduler	<code>PersistJobService</code> every 15 minutes, persisted across reboots
4	Broadcast receiver	5 intent actions: <code>BOOT_COMPLETED</code> , <code>LOCKED_BOOT_COMPLETED</code> , <code>MY_PACKAGE_REPLACED</code> , <code>USER_PRESENT</code> , custom WATCHDOG
5	Native watchdog	<code>libwatchdog.so</code> (PIE executable despite the extension); scans <code>/proc/*/cmdline</code> every 120s, reinstalls from cache if bot is missing
6	Shell watchdog	<code>.w</code> script implementing the same 120-second loop
7	Root boot scripts	Magisk (<code>post-fs-data.d</code> , <code>service.d</code>) and KernelSU (<code>ksu/post-mount.d</code>)
8	OOM lock + anti-uninstall	<code>oom_score_adj -1000</code> (plot armor, but for processes), blocks force-stop and uninstall from Settings, disables phantom process killer
9	ADB port hijack	Binds <code>0.0.0.0:5555</code> , kills <code>adbd</code> with 10 commands

The ADB hijack deserves a note. Where Katana remaps the ADB port to 12341 and keeps it available for its own use, CECbot simply closes the door. And where Katana has a dead-man's-switch self-destruct that cleans up after itself if C2 goes silent, CECbot has no such courtesy. Once installed, it stays.

OEM-specific evasion

On rooted devices, CECbot applies vendor-specific commands to bypass battery optimization and background execution restrictions for 7 Android manufacturers:

vendor	technique
Xiaomi/Redmi/POCO	MIUI permission editor, <code>force_miui_destroy 0</code> , <code>AUTO_START allow</code>

vendor	technique
Huawei/Honor	<code>RUN_IN_BACKGROUND allow, hwPfm setpkg -1</code>
Samsung	<code>except-idle-whitelist, sleeping_apps_excluded</code>
OPPO/Realme	<code>AUTO_START allow, BOOT_COMPLETED allow</code>
Vivo	<code>AUTO_START allow</code>
OnePlus	<code>RUN_IN_BACKGROUND allow, RUN_ANY_IN_BACKGROUND allow</code>
All (rooted)	Doze whitelist, standby bucket <code>active</code> , battery stats reset

Katana, as a Mirai fork, has no awareness of the Android framework at all. CECbot speaks each vendor's language, and uses it to ensure the platform's own power management will never interfere. The vendor-specific persistence is a localization effort most legitimate developers would not ship.

TV box takeover

The bot detects Android TV via `android.software.leanback`, `android.hardware.type.television`, or devices lacking both telephony and touchscreen. On detection, it executes a comprehensive takeover:

Launcher hijack. Sets itself as the default home activity (`cmd package set-home-activity <pkg>/MainActivity`). The user sees a blank screen. The remote control does nothing useful.

SELinux bypass. `setenforce 0, setprop ro.boot.selinux permissive`.

OTA sabotage. Disables 10 manufacturer OTA updater packages spanning 6 chipset vendors (Amlogic, Rockchip, Allwinner, HiSilicon, ZTE, MediaTek), blocks automatic OTA updates, disables auto-time sync, and sinkholes 5 OTA domains via `/etc/hosts`:

```
127.0.0.1 ota.amlogic.com
127.0.0.1 update.amlogic.com
127.0.0.1 ota.rockchip.com
127.0.0.1 update.rockchip.com
127.0.0.1 ota.allwinnertech.com
```



The infected device will never receive a firmware update again. For devices still receiving OTA updates, this is a meaningful loss. For the many that were already end-of-life, the effect is academic.

Package verification bypass. Disables `package_verifier_enable` and `verifier_verify_adb_installs`, preventing Google Play Protect or sideload scanning from flagging the APK.

Device profiling. Command `0x44` checks for 33 streaming app package names (Netflix, YouTube TV, Disney+, HBO Max, Hulu, Twitch, Kodi, Plex, and 25 others) and reports the results to the C2. Whether this is used for sandbox detection, bot quality assessment, or device classification is not clear from the binary alone. Checking 33 streaming packages before taking over the device is, in its way, due diligence.

What the user doesn't see

So: a DDoS bot with Signal-grade cryptography, 9 layers of persistence, vendor-specific evasion for 7 manufacturers, and the ability to permanently disable firmware updates on 6 chipsets. That is already a considerable upgrade from the Mirai fork it replaced. But none of that is why we named it CECbot.

HDMI Consumer Electronics Control is a one-wire protocol on HDMI pin 13 that allows connected devices to send commands to each other: turning TVs on and off, adjusting volume, switching inputs. It is supported by virtually every TV manufactured in the last 15 years, marketed under names that reveal nothing about the underlying protocol (Samsung Anynet+, LG SimpLink, Sony Bravia Sync). CEC is also [notorious](#) for unreliable interoperability across device vendors, which makes a botnet operator betting on it an interesting design choice. Debugging CEC across a heterogeneous fleet of cheap TV boxes is, in its own way, a form of accidental tech support. But for a single standby command between two directly connected devices, reliability is not the problem.

CECbot is, to our knowledge, the first documented malware to weaponize HDMI-CEC in the wild. Prior work on CEC as an attack vector exists only in a [2021 doctoral dissertation](#) (Puche Rondon, Florida International University). We are not aware of any real-world malware that has previously used this technique. A practical caveat: not all Android versions expose CEC APIs or ship `cec-client`, so the CEC capabilities will be inert on a subset of the target device population. The code accounts for this with multiple fallback paths (`cec-client`, `cmd hdmi_control cec_send`, direct `keyevent`), but on devices where none are available, the TV stays on and the operator is limited to the Android power key.

One plausible operational rationale: CECbot replaces the home launcher with its own blank `MainActivity` (see). Without CEC standby, the next time the TV is on, the screen is blank, the remote is unresponsive, and the interface the user relied on to launch Netflix is gone. That prompts investigation. With CEC standby, the TV appears to be off — and on a cheap set-top box in a bedroom or guest room, a TV that seems to have turned itself off is unremarkable. The CEC standby would not prevent discovery, but it would delay it.

We have not observed C2 traffic to confirm this usage pattern, and the CEC commands are not triggered automatically (see below). But the capability and the launcher hijack exist in the same binary, and the combination is suggestive.

Four commands provide full HDMI bus control, not just standby:

1. **Standby** (command `0x23`): sends both `input keyevent 26` (Android power key) and `echo standby 0 | cec-client -s -d 1` (CEC standby). Belt and suspenders: the Android keyevent turns off the local display, the CEC command tells the TV to sleep.
2. **Bus enumeration** (command `0x40`): CEC bus scan (`echo scan | cec-client`) plus Amlogic HDMI hotplug state and `dumpsys hdmi_control`. Discovers all HDMI-connected devices and checks whether a TV is actually present. There is no point in whispering "sleep" into an empty room.
3. **Arbitrary CEC frame injection** (command `0x41`): sends raw CEC frames via `echo tx <hex> | cec-client`, with fallback to `cmd hdmi_control cec_send`. This is the most significant capability: the operator provides raw hex bytes, enabling any CEC opcode. Beyond standby, this includes waking the TV (`Image View On`), hijacking the active HDMI input (`Active Source`, `Routing Change`), displaying arbitrary text on the TV screen (`Set OSD String`), simulating remote control buttons, and controlling connected soundbars or AV receivers.
4. **Display info** (command `0x46`): reads display DPI, refresh rate, HDR capability, and HDMI configuration.

A separate user activity check (command `0x45`) reports whether the screen is on, the device is locked, and whether music is playing. This could inform CEC standby decisions, though we have not observed C2 traffic to confirm whether the operator chains these commands in practice.

An important nuance: none of the CEC capabilities are triggered automatically during DDoS attacks. There is no code linking the attack engine to CEC standby. Every CEC operation requires an explicit command from the C2. The capability is there; whether and how the operator uses it is an operational question we cannot answer from the binary alone.

What the operator sees

Whether or not the operator uses CEC standby, the bot gives them full visibility into the home network it sits on.

The LAN discovery command (`0x53`) performs an ICMP sweep of the local /24 subnet, correlates live hosts against the ARP table, and returns a list of every reachable device with its MAC address. Combined with the network profiling command (`0x52`, which collects the local IP, gateway, DNS servers, WiFi SSID, and signal strength) and the port scanner (`0x50`), the operator can map the victim's entire home network in seconds: what devices are present, what services they run, and how to reach them.

This is a capability that Katana does not have. Katana is a DDoS bot that happens to run on an Android TV box. CECbot treats the Android TV box as what it is: a general-purpose computer sitting on the local network, on a trusted subnet. The DDoS and proxy capabilities monetize the device's bandwidth and IP address. The LAN discovery capabilities treat it as a vantage point.

Synthient's [Kimwolf research](#) documented the risk of compromised Android TV devices serving as pivot points into internal networks. Kimwolf itself does not perform automated LAN scanning. CECbot does, with purpose-built commands that return structured results to the C2.

The two headline capabilities are complementary. CEC standby ensures the user does not notice anything wrong. LAN discovery ensures the operator knows everything about the network the user thinks is private. The television is off. The lights, behind it, are very much on.

This is what makes CECbot architecturally coherent in a way that Katana is not. Katana fights the platform: it compiles kernel modules, overwrites system partitions, and runs as a raw ELF binary with no framework awareness. CECbot works *with* the platform. It uses Android's own scheduling to persist, Android's own power management APIs to stay running, Android's own package manager to block uninstallation, HDMI's own control protocol to hide the evidence, and the network stack to map the home. The operating environment is not an obstacle to be overcome. It is a tool to be used.

Katana brute-forces the platform. CECbot RTFM'd the entire Android SDK.

Detection

Network signatures

- TCP segments to C2 begin with `\xBE\xEF` magic
- Handshake: `\xBE\xEF\x00` + exactly 64 bytes (client) or 96 bytes (server)
- Default C2 port: 10213/TCP
- DNS queries for `sdkconnector[.]com` or `sdkconnect121[.]st`

SMTP relay hostnames (strong indicator)

Attack 7's SMTP sub-type uses placeholder hostnames as EHLO/HELO arguments: `mail.server.local`, `smtp.domain.net`, `mx1.host.org`, `relay.mail.com`. These are identical to the Katana SMTP payloads, inherited or copied despite the complete codebase rewrite. None of these hostnames resolve; `mail.server.local` uses the `.local` TLD reserved for mDNS and cannot resolve via standard DNS. They remain the strongest single network indicator for both Katana and CECbot attack traffic.

Indicators of compromise

Machine-readable IoC files are in [iocs/](#):

File	Contents
domains.csv	C2 domains including Tor fallback
ips.csv	C2 IPs and fast-flux relay nodes
hashes.csv	APK SHA-256 hash
keys.csv	Ed25519 server public key, XOR key, Tor response key

References

- [Katana report](#) — predecessor botnet by the same operator (Nokia Deepfield ERT, March 2026)
- [Aisuru ecosystem report](#) — documents the broader ecosystem including Jackskid's competition with this operator (Nokia Deepfield ERT, March 2026)
- Puche Rondon, "[Novel Attacks and Defenses for Enterprise IoT](#)" (doctoral dissertation, Florida International University, 2021) — prior academic work on CEC as an attack vector