WHAT_IS_THIS.md

> Analysis of source code extracted from a Docker container associated with the **JCO4032** router. The image was published as `yskp/jio-jco4032-new:latest` on Docker Hub.

TL;DR

This is the source code for **Cyberiot** (also branded **cyb3riot** / **Netonomy**) — a third-party **router-level network surveillance and remote packet-blocking agent**. It is **not part of the router's stock firmware**. It is an add-on installed on the device that:

1. Hooks into the Linux kernel's netfilter stack to inspect every packet the router forwards.
2. Drops packets to/from IPs and MAC addresses chosen by a remote server.
3. Continuously uploads connection metadata, LAN device inventory (MAC, IP, hostname), and detected "threats" to `api.netonomy.co`.

Originally written for the **Netgear DGN2200v2** (per the repo's `README.md`), then cross-compiled to MIPS for additional router platforms — JCO4032 appears to be one such MIPS-Linux target.

Provenance

Signal	Value
Build container	`yskp/buildroot-2010.02-dev`, `yskp/buildroot-2016.08-3.4.11-rt19`
Target arch	MIPS (big-endian, uClibc), Linux 2.6.23 / 2.6.30 / 3.4.11
Production C&C	`https://api.netonomy.co:443` (`agent/bin/cyberiot.cfg.production_server`)
Deployment server	`http://api.cyb3riot.com/v1/routerAgent` (`deploy`)
Kernel module license	`MODULE_LICENSE("Proprietary")` (`module/src/nfmodule.c`)
Original target router	Netgear DGN2200v2 (per `README.md`)
Build system	Atlassian Bitbucket Pipelines (`bitbucket-pipelines.yml`)

The vendor "Cyberiot / Netonomy" was an IoT-security startup (~2016–2017) that pitched router-resident smart-home protection. The domains are still wired into the production config shipped on the device.

High-Level Architecture

Two components, one upstream service.

Component 1: Kernel module (`module/`)

Output artifact: `nfmodule.ko`

Netfilter hook (`module/src/nfmodule.c`)

- Registers `hook_func` at hook number `0` on protocol family `0x7000000` (a hand-rolled non-standard value the code comments call "PF_BRIDGE … so that we see local packets too"), priority `INT_MIN`.
- For every Ethernet frame:
  1. `report_connection_if_should(...)` — logs interesting TCP packets (SYN / FIN / RST or every ~1 MB of sequence space) into a cyclic buffer for userspace pickup.
  2. `blockedMacCollection->ProcessPacket(...)` → returns `BLOCK` ⇒ `NF_DROP`.
  3. `threatCollection->ProcessPacket(...)` → returns `BLOCK` ⇒ `NF_DROP`.

`/proc/cyberiot` interface

- **Read** (`procfile_read`) — userspace pulls a stream of `struct event_report` (blocked-IP hit events).
- **Write** (`procfile_write`) — userspace pushes a `|`-separated list of dotted-quad IPs; each becomes a blocked address in the kernel's `ThreatCollection`.

The module also wires `procfile_read` / `procfile_write` into the proc entry via a raw pointer arithmetic trick (`proc_file_ptr + 0x10`, `+ 0x11`) — a brittle workaround that depends on the in-memory layout of `struct proc_dir_entry` for a specific kernel build. This is why `elfpatch.c` exists (see "Notable hacks" below).

`/dev` char device (`module/src/char_device.c`)

A character device used to stream `struct reported_connection` records (TCP flow metadata: protocol, LAN direction, flags, seq, payload size, src/dst MAC, src/dst IP, src/dst port, timestamp — see `common/traffic_report.h`) from the kernel to the userspace agent.

Threat & block storage

- `ThreatCollection` — fixed-size table of IPv4 addresses to drop, with per-entry timestamps for stale-entry expiry (default 1 hour) and per-entry reporting throttle (default 30,000 seconds).

- `BlockedMacCollection` — same idea, but matched on Ethernet source/dest MAC.
- `ThreatEventLog` — cyclic buffer of "we blocked address X" events, drained via `/proc/cyberiot`.

Module configuration parameters

```
report_interval_msec      (default 30000000)  -- block-event re-report throttle
stale_address_timeout_sec (default 3600)      -- block-list TTL
```

Component 2: Userspace agent (`agent/`)

Output artifact: `controller` (MIPS) and a debug `controller_debug` build that mocks the kernel module.

Main loop (`agent/src/main.c`)

Every 5 seconds:

```
cyberiotKnownThreatPump   ->PumpKnownThreats(...)        // pull blocklist
cyberiotTrafficPump       ->PumpReportedNetworkTraffic() // push flow metadata
cyberiotDetectedThreatPump->PumpDetectedThreats(...)     // push block events
synchronizingThreatCollection->SynchronizeThreats(...)   // reconcile kernel <-> server
synchronizingThreatCollection->CleanupOldThreats(...)
cyberiotDevicePump        ->PumpDevices(...)             // push LAN device list (hourly)
```

Pumps in detail

Pump	Direction	Endpoint	Payload
`CyberiotKnownThreatPump`	server → router	`GET /v1/threats`	JSON list of IPs to start dropping
`CyberiotTrafficPump`	router → server	`POST /v1/traffic`	Per-connection JSON: src/dst IP+port, MACs, direction, SYN/FIN/RST, payload size, timestamp
`CyberiotDetectedThreatPump`	router → server	`POST /v1/blockedThreats`	Per-block event read from `/proc/cyberiot`
`CyberiotDevicePump`	router → server	`POST /v1/devices`	Inventory of every LAN device (MAC + IP + hostname) read from ARP cache + DHCP leases, every hour

Authentication (`agent/src/CyberiotServer.c`)

On startup the agent calls `POST /v1/authenticate/router` with:

```
{
  "macWifi": ["AA:BB:CC:...", ...],   // every non-loopback MAC on the device
  "routerId": "<hardcoded 24-char ID from run script>",
  "version": "<git commit ID baked in at build>"
}
```

…and receives a bearer token used for every subsequent request.

The shipped `run` script hardcodes two router IDs (one for the local-dev server, one for prod), suggesting either dev sample IDs or that all devices in the field originally shared a single tenant identity.

Data sources read on the router

- `/proc/net/arp` — ARP cache → who is on the LAN (`ArpCacheReader.c`)
- DHCP lease file — LAN device hostnames (`DhcpCacheReader.c`)
- `/proc/net/ip_conntrack` — full connection-tracking table (`conntrack_parser.c`)
- Kernel module's `/dev` char device — live flow metadata
- Kernel module's `/proc/cyberiot` — block-event log
- `ioctl(SIOCGIFCONF / SIOCGIFHWADDR)` — every network interface and its MAC

Vendored dependencies

The agent statically links a fairly large amount of vendored code:

- **mbedtls** — TLS for HTTPS to `api.netonomy.co`
- **http-parser** — HTTP/1.x response parsing
- **uriparser** — URL parsing
- **jsmn** — minimal JSON tokenizer
- A custom `Reader` / `Writer` abstraction layer over files, sockets, and stdout
- A small custom DNS resolver (`net/resolv.h`)

A pinned set of TLS root certificates is compiled into `ReleaseCertificates.c` .

Build & deployment pipeline

`build_and_test`

For each platform variant:

1. Build `nfmodule.ko` (MIPS), run kernel-module unit tests (x86 host build).
2. **Run `elfpatch` on the `.ko`** (see below).
3. Build the userspace agent (x86 for tests, MIPS for shipping).
4. Run unit tests, integration tests, valgrind leak check.

`deploy`

Pushes the three artifacts to the vendor's deployment service:

```
wget --post-file=agent/bin/controller                http://api.cyb3riot.com/v1/routerAgent?type=user&buildConfig=...
wget --post-file=module/nfmodule.ko                  http://api.cyb3riot.com/v1/routerAgent?type=kernel&buildConfig=...
wget --post-file=agent/bin/cyberiot.cfg.production_server http://api.cyb3riot.com/v1/routerAgent?type=config&buildConfig=...
```

Auth via two env-var headers (`DEPLOYMENT_URL_SECRET_KEY` / `_VALUE`). No HTTPS for the deploy step — the artifacts including the signed kernel module are uploaded over plain HTTP.

Supported build configurations

From `bitbucket-pipelines.yml` :

Build config	Kernel	Notes
`NO_CONNTRACK_LINUX_020623_UP`	2.6.23 UP	No `/proc/net/ip_conntrack`
`CONNTRACK_LINUX_020630_SMP`	2.6.30 SMP	
`CONNTRACK_LINUX_030411_SMP`	3.4.11 SMP	

Per `README.md` , the server side maps each router model to one of these `buildConfig` s; JCO4032 maps to whichever one matches its kernel.

Notable hacks worth calling out

`elfpatch.c` — relocation rewriter

After the kernel module is compiled, `elfpatch` opens the `.ko` , finds the `.rel.gnu.linkonce.this_module` section, and **rewrites the second relocation's `r_offset`** to a platform-specific value (`cleanupOffset` in `platform.c` — `0x114` or `0x118` depending on target kernel).

Effect: redirects the module's `cleanup_module` slot in `struct module` so the same compiled binary loads into kernels whose `struct module` layout differs from the one the toolchain was built against. This is the same technique used to make a hand-crafted `.ko` load into a "foreign" kernel, and is a strong indicator that the module is being deliberately shoehorned into kernels it was not compiled to match.

`proc_file_ptr + 0x10` write

`nfmodule.c` casts `struct proc_dir_entry *` to `void**` and stores function pointers at offsets `+0x10` and `+0x11` to register the read/write handlers, bypassing the documented `proc_dir_entry->read_proc` / `->write_proc` fields. Same motive: target a specific in-memory layout without having matching kernel headers.

Non-standard netfilter family

`nfho.pf = 0x7000000` — not a real `PF_*` constant. The comment claims this is "PF_BRIDGE rather than PF_INET, so that we see local packets too." This is set on a `nf_hook_ops` and then registered with `nf_register_hook`, which would normally require a real protocol family. The value is little-endian `0x07000000` ⇒ presumably the on-the-wire NFPROTO_BRIDGE constant after endian games, but the route taken is unusual.

Hardcoded router IDs

The `run` script ships with two specific 24-character router IDs baked in (one for local, one for prod). On a production device these likely come from somewhere else, but the shipped artifact includes them.

Capability summary

What the device can do once this is installed	Code reference
See every TCP/IP packet through the router	`module/src/nfmodule.c:31`
Drop packets to/from server-supplied IPv4s	`module/src/ThreatCollection.c`
Drop packets from server-supplied MAC addresses	`module/src/BlockedMacCollection.c`
Upload TCP flow metadata (src/dst IP+port, MACs, flags, payload sizes, timestamps)	`agent/src/CyberiotTrafficPump.c`, `common/traffic_report.h`
Upload list of every LAN device (MAC, IP, hostname) hourly	`agent/src/CyberiotDevicePump.c`
Upload every "block" event with the destination IP it dropped	`agent/src/CyberiotDetectedThreatPump.c`
Receive new blocklists at any time from the server	`agent/src/CyberiotKnownThreatPump.c`
Authenticate with the vendor using the router's WiFi MAC addresses	`agent/src/CyberiotServer.c`

Security & privacy assessment

What this is, in plain terms

A **vendor-controlled remote packet filter and traffic-telemetry agent** sitting in your router's data path. Whoever controls `api.netonomy.co` can, at any moment:

- Tell your router to drop traffic to or from any IPv4 address.
- Tell your router to drop all traffic from any MAC address on your LAN.
- See every TCP flow your devices open, by 5-LAN-tuple plus MAC addresses on both ends.
- See every device on your LAN, by MAC, IP, and hostname.

This is the same capability profile as commercial "router security" products (Bitdefender Box, F-Secure SENSE, etc.). The legitimate framing is "we block known malicious IPs and warn you about compromised IoT devices." The technical capability is indistinguishable from a network surveillance and remote-kill-switch agent.

Specific red flags in the implementation

1. **Vendor (Cyberiot / Netonomy) appears defunct.** The agent is still configured to dial out to `api.netonomy.co`. If that domain is ever re-registered by anyone else, that party inherits the ability to push blocklists to every JCO4032 still running this firmware.
2. **Deployment over plain HTTP.** The signed kernel module, agent binary, and production config are uploaded to `api.cyb3riot.com` over unencrypted HTTP at build time (`deploy:14-16`).
3. **Embedded TLS root store.** The agent ships with its own pinned CA list (`ReleaseCertificates.c`) — server connections do not use the device's system trust store, so the user has no way to revoke trust without reflashing.
4. **Hardcoded `routerId`s in the shipped `run` script.** Indicates this build was at least at some point operating with shared/tenant-wide identities.
5. **Kernel-module loaded via relocation rewriting** rather than being compiled for the exact kernel. The same `.ko` can be reused across kernel versions, which is convenient for the vendor and makes the install indistinguishable in structure from a rootkit's loader trick.
6. **No user-facing opt-out is present in this source.** The agent runs continuously, authenticates on startup, and there is no kill-switch in the userspace `main()` other than process death.

Mitigations if you own this router

- Block outbound traffic from the router itself to `api.netonomy.co` and `api.cyb3riot.com` at your upstream / ISP-side firewall, if available.
- Inspect the router's running processes (look for `controller`) and loaded modules (look for `nfmodule`).
- If your ISP provided this device, you should know that **all your LAN-side connection metadata is being shipped to a third party** by default. Whether you opted into that is a question for the ISP's terms of service.
- Consider replacing the device with router firmware you control (OpenWrt etc., if hardware allows).

Repository layout

```
.
├── README.md                      -- Build instructions (mentions DGN2200v2)
├── bitbucket-pipelines.yml        -- CI build matrix (3 kernel variants)
├── build_and_test                 -- Top-level build + test driver
├── deploy                         -- Pushes built artifacts to api.cyb3riot.com
├── elfpatch.c                     -- ELF relocation rewriter for the .ko
├── platform.c / platform.h        -- Per-kernel cleanupOffset constants
├── common/
│   ├── git_commit.h               -- Git SHA baked into auth payload
│   └── traffic_report.h           -- Wire format for kernel->userspace flow records
├── module/                        -- Linux kernel module (nfmodule.ko)
│   ├── Makefile, Makefile.tests
│   ├── build, netgearBuild        -- Cross-compile scripts (MIPS / uClibc)
│   └── src/
│       ├── nfmodule.c             -- Netfilter hook + /proc + module init/exit
│       ├── ThreatCollection.{c,h} -- Blocked-IP table
│       ├── BlockedMacCollection.* -- Blocked-MAC table
│       ├── ThreatEventLog.*       -- Block-event ring buffer
│       ├── PacketProcessor.*      -- Per-packet decision logic
│       ├── connection_reporting.* -- TCP flow metadata capture
│       ├── cyclic_buffer.*        -- Lockless ring buffer
│       ├── char_device.*          -- /dev character device for flow records
│       ├── platform.{c,h}         -- Per-kernel offsets
│       └── ...
├── agent/                         -- Userspace controller (controller)
│   ├── Makefile                   -- ARCH=x86 (tests) or ARCH=mips (ship)
│   ├── bin/
│   │   ├── run
│   │   ├── cyberiot.cfg.production_server  -- https api.netonomy.co 443
│   │   └── cyberiot.cfg.local_server       -- http 192.168.81.1 8080
│   ├── src/
│   │   ├── main.c                          -- 5-second pump loop
│   │   ├── CyberiotServer.{c,h}            -- Auth
│   │   ├── CyberiotKnownThreatPump.*       -- Pull blocklist
│   │   ├── CyberiotTrafficPump.*           -- Push flow metadata
│   │   ├── CyberiotDetectedThreatPump.*    -- Push block events
│   │   ├── CyberiotDevicePump.*            -- Push LAN device inventory
│   │   ├── SynchronizingThreatCollection.* -- Userspace<->kernel sync
│   │   ├── DeviceMapBuilder.*              -- ARP + DHCP -> device list
│   │   ├── ArpCacheReader.*, DhcpCacheReader.*
│   │   ├── conntrack_parser.*              -- /proc/net/ip_conntrack reader
│   │   ├── StatelessHttpConnection.*       -- TLS HTTP client (mbedtls)
│   │   ├── ReleaseCertificates.*           -- Pinned root CAs
│   │   ├── cyberiot_config.*               -- Config file + DNS resolution
│   │   ├── netutils.*                      -- MAC enumeration
│   │   ├── mbedtls/                        -- vendored TLS
│   │   ├── http-parser/                    -- vendored HTTP parser
│   │   ├── uriparser/                      -- vendored URL parser
│   │   ├── jsmn.*                          -- vendored JSON tokenizer
│   │   └── ...
│   └── tests/                              -- Unit + integration tests
└── tools/
    └── hexdump.c
```

Bottom line

It's a **vendor-installed network surveillance and remote-blocking agent** running inside the JCO4032's firmware, marketed under the "Cyberiot / Netonomy" brand. It reports every connection on the LAN to a third party and accepts remote commands to drop traffic. Whether that constitutes "security" or "spyware" depends on what the device's owner was told when it was provisioned — but it is definitely not standard router firmware code, and the upstream vendor's infrastructure appears to be

inactive, which makes the dial-home channel a latent risk on its own.