بسمه‌تعالی

بهره‌برداری از آسیب‌پذیری تزریق فرمان در روتر TP-Link Archer A7

گزارش آسیب‌پذیری

در طی مسابقات Pwn2Own توکیو در پاییز گذشته، اشخاصی به نام Pedro Ribeiro و Radek Domanski از آسیب‌پذیری تزریق فرمان به عنوان بخشی از زنجیره بهره‌برداری برای اجرای کد بر روی روتر بی سیم TP-Link Archer A7 استفاده کردند که ۵۰۰۰ دلار برای آن‌ها درآمد داشت. باگ به کار رفته در این بهره‌برداری اخیراً وصله شده است. در این گزارش به بررسی آسیب‌پذیری تزریق فرمان که در نوامبر ۲۰۱۹ منتشر شد می‌پردازیم.

این آسیب‌پذیری در سرور tdp daemon (/usr/bin/tdpServer) در روتر TP- Link Archer A7 (AC1750) با نسخه سخت‌افزار ۵، معماری MIPS و سیستم‌عامل نسخه 190726 وجود دارد که توسط مهاجمی در سمت LAN روتر قابل بهره‌برداری است و نیاز به احراز هویت ندارد.

پس از بهره‌برداری مهاجم قادر است هر دستوری شامل بارگیری و اجرای یک دودویی از یک میزبان دیگر را با دسترسی root اجرا کند. این آسیب پذیری با نام CVE-2020-10882 شناخته می‌شود و توسط TP-Link با نسخه سیستم‌عامل A7 (US) _V5_200220 رفع شده است. در کلیه قطعه کدها و توابع در این گزارش از /usr / bin / tdpServer، نسخه سیستم عامل ۱۹۰۷۲۶ استفاده شده است.

۱ جزئیات tdpServer

سرور tdp daemon به پورت ۲۰۰۰۲ UDP بر روی خط اتصال ۰.۰.۰.۰ گوش می‌دهد. عملکرد کلی daemon در این مرحله توسط نویسندگان کاملاً درک نشده است، زیرا این مورد برای بهره‌برداری غیر ضروری بود. با این حال به نظر می‌رسد daemon پلی بین برنامه‌های موبایلی و روتر TP-Link، با امکان ایجاد نوعی کانال کنترل از برنامه موبایل است. سرور daemon با برنامه‌های موبایل از طریق بسته‌های UDP با محتوای رمزگذاری شده ارتباط برقرار می‌کند. قالب بسته معکوس شده و در شکل ۱ قابل مشاهده است.

```
#define PACKET_SZ 0x400
#define PACKET_HDR_SZ 0x10
#define PAYLOAD_SZ (PACKET_SZ - PACKET_HDR_SZ)


typedef struct tpdp_packet {
    // packet version, fixed to 1
    uint8_t version;

    // packet type; tdpd == 0 or onemesh == 0xf0
    uint8_t type;

    // onemesh opcode, used by the onemesh_main switch table
    uint16_t opcode;

    // packet length
    uint16_t len;

    // some flag, has to be 1 to enter the vulnerable onemesh function
    uint8_t flags;

    // dunno what this is
    uint8_t unknown;

    // sn == serial number ? can be any value
    uint32_t sn;

    // packet checksum
    uint32_t checksum;

    // the payload can have up to 0x3F0 bytes
    uint8_t payload[PACKET_SZ-PACKET_HDR_SZ];
} packet;
```

شکل ۱ : قالب معکوس بسته‌های tdpServer

نوع بسته تعیین می‌کند چه سرویسی در Daemon مورد استفاده قرار خواهد گرفت. نوع ۱ باعث می‌شود Daemon از سرویس tdpd استفاده کند، که بسته‌ها را به آسانی با یک مقدار hash مشخص TETHER_KEY پاسخ می‌دهد. این مسئله به آسیب‌پذیری مربوط نیست، بنابراین جزییات آن ذکر نشده است.

نوع دیگر بسته‌ها 0xf0 است که از سرویس onemesh استفاده می‌کند؛ در این سرویس آسیب‌پذیری نهفته است. به نظر می‌رسد سرویس onemesh یک فناوری mesh اختصاصی است که توسط TP-Link در نسخه‌های سیستم‌عامل اخیر تعدادی از روترهای آن‌ها معرفی شده است.

۲ بررسی آسیب‌پذیری

هنگامی که دستگاه شروع به کار می‌کند، اولین تابعی که فراخوانی می‌شود (tdpd_pkt_handler_loop (آدرس 0x40d164) است که یک سوکت UDP باز کرده و به پورت ۲۰۰۰۲ گوش می‌دهد. هنگامی که بسته دریافت شد، این تابع بسته را به (tpdp_pkt_parser (0x40cfe0) تحویل می‌دهد. یک قطعه از کد این تابع در شکل ۲ نشان داده شده است:

```
int tdpd_pkt_parser(int packet,int packet_len,int packet_reply,int *packet_reply_len,int param_5)

{
(...)
    if (packet != 0) {
        packet_len = return_0x10();
        if (packet_sz < packet_len) {
            print_debug("tdpdServer.c:709","recvbuf length = %d, less than hdr\'s 16",packet_sz);
            return 0xffffffff;
        }
        packet_len = tdpd_get_pkt_len(packet);
        if (packet_len < 1) {
            pcVar7 = "tdpdServer.c:716";
            pcVar8 = "tdp pkt is too big";
        }
        else {
            print_debug("tdpdServer.c:719","tdp pkt length is %d",packet_len);
            packet_len = tdpd_pkt_sanity_checks(packet,packet_len);
            if (packet_len < 0) {
                return 0xffffffff;
            }
(...)
```

شکل ۲: بخش اول tpdp_pkt_parser()

در اولین قطعه، تجزیه کننده ابتدا بررسی می‌کند که اندازه بسته گزارش شده توسط سوکت UDP حداقل 0x10 (به اندازه header) باشد. سپس (tdpd_get_pkt_len (0x40d620) را فراخوانی می‌کند. این تابع مقدار طول بسته‌های مشخص شده مطابق با header بسته‌ها (فیلد len) را برمی‌گرداند. اگر طول بسته از 0x410 تجاوز کند، تابع مقدار ۱- را برمی‌گرداند.

بررسی نهایی توسط (tdpd_pkt_sanity_checks (0x40c9d0) انجام می‌شود، که به علت اختصار نشان داده نمی‌شود، اما دو تأیید انجام می‌دهد: ابتدا بررسی می‌کند نسخه بسته (فیلد نسخه، اولین بایت در بسته) برابر با ۱ باشد. سپس میزان checksum بسته را با استفاده از تابع checksum محاسبه می‌کند: (tpdp_pkt_calc_checksum (0x4037f0).

برای فهم بهتر، در شکل ۳ تابع (calc_checksum شرح داده شده است که بخشی از کد بهره‌برداری lao_bomb می‌باشد. این تابع به جای (tpdp_pkt_calc_checksum نشان داده شده تا فهم آن آسان‌تر گردد.

```
void calc_checksum(packet *pkt, int len) { uint8_t *ptr;
    uint8_t val;
    uint32_t res;
    int32_t i = 0;

    // set magic var before calculating checksum
    pkt->checksum = 0x5a6b7c8d;

    res = 0xffffffff;
    ptr = (uint8_t*)pkt;

    while (i < len) {
        val = *(uint8_t*)ptr;
        res = *(uint32_t *)(reference_tbl + (((uint32_t)val ^ res) & 0xff) * 4) ^ (res >> 8);
        ptr += 1;
        i += 1;
    }

    res = ~res;
    pkt->checksum = res;
    return;
}
```

شکل ۳: تابع calc_checksum() از کد بهره‌برداری lao_bomb

محاسبه checksum کاملاً ساده است. در ابتدا باید مقدار متغیر 0x5a6b7c8d در فیلد بسته checksum تنظیم شود، سپس از جدول reference_tbl (جدولی با ۱۰۲۴ بایت) برای محاسبه checksum در کل بسته شامل header استفاده می‌کند. پس از تایید صحت checksum، تابع tdpd_pkt_sanity_checks() مقدار ۰ را بازمی‌گرداند. سپس بخش بعدی tdpd_pkt_parser() را فراخوانی می‌کنیم. در شکل ۴ این عملیات قابل مشاهده است.

```
(...)
    if (*(char *)(packet + 1) == '\0') {
        (...)
    }

    if ((*(char *)(packet + 1) == 0xf0) && (onemesh_flag == '\x01')) {
        ret = onemesh_main(packet,packet_sz,packet_reply,packet_reply_len);
        return ret;
    }
(...)
```

شکل ۴ : بخش دوم tdpd_pkt_parser()

در اینجا بررسی می‌شود مقدار دومین بایت بسته (فیلد نوع)، ۰ (tdpd) یا 0xf0 (onemesh) است. در بخش دوم، همچنین بررسی می‌شود که متغیر جهانی onemesh_flag طبق پیش‌فرض برابر ۱ تنظیم شده باشد. سپس onemesh_main() (0x40cd78) را فراخوانی می‌کنیم.

تابع onemesh_main() در اینجا به علت اختصار نمایش داده نمی‌شود، اما وظیفه آن فراخوانی تابع دیگری مبتنی بر فیلد opcode بسته است. به منظور یافتن تابع آسیب‌پذیر، مقدار فیلد opcode برابر ۶ و فیلدهای flag برابر ۱ قرار می‌گیرند. در این مثال، onemesh_slave_key_offer() (0x414d14) فراخوانی می‌شود.

شکل ۵ نمایانگر تابع آسیب‌پذیر است، از آنجا که این تابع بسیار طولانی است، فقط بخش‌های مرتبط نمایش داده شده‌اند.

```
int onemesh_slave_key_offer(int packet,undefined4 packet_sz,undefined *packet_reply,int *packet_r eply_len)
{
(...)
    if (((packet == 0) || (packet_reply == NULL)) || (packet_reply_len == NULL)) {
        __s = "tdpOneMesh.c:2887";
        __dest = "Invalid parameters";
        goto return_-1_n_exit;
    }
    payload_dec = (void *)(packet + 0x10);
    memset(plaintext_out,0,0x400);
    ret = tpapp_aes_decrypt(payload_dec,(uint)*(ushort *)(packet + 4),"TPONEMESH_Kf!xn?gj6pMAt-wBNV_TDP",plaintext_out,0x400);
    if (ret != 0) {
        __s = "tdpOneMesh.c:2896";
        __dest = "Failed to decrypt.";
        goto return_-1_n_exit;
    }
    __n = strlen(plaintext_out);
    print_debug("tdpOneMesh.c:2899","plainLen is %d, plainText is %s",__n,plaintext_out);
(...)
```

شکل ۵: بخش اول onemesh_slave_key_offer()

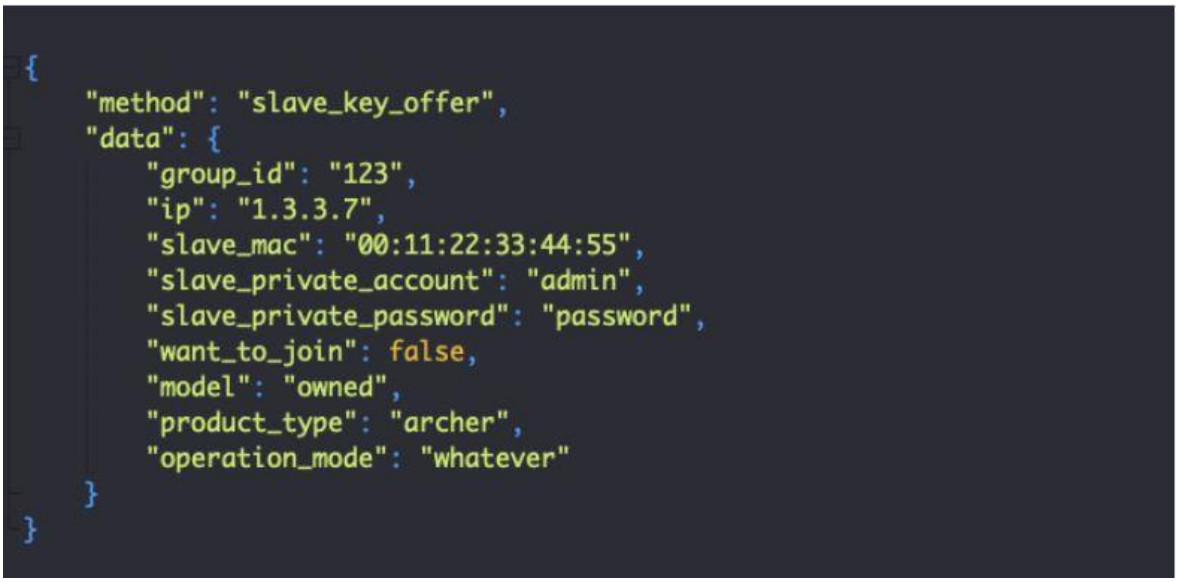
در بخش اول onemesh_slave_key_offer()، این تابع بسته را به tpapp_aes_decrypt() (0x40b190) تحویل می‌دهد. تابع tpapp_aes_decrypt() نیز به علت اختصار نمایش داده نمی‌شود، اما فهم آن ساده است و وظیفه آن رمزگشایی بسته با استفاده از الگوریتم AES و کد ایستای PONEMESH_Kf!xn?gj6pMAt-wBNV_TDP می‌باشد.

فرض می‌شود که tpapp_aes_decrypt قادر به رمزگشایی کامل بسته باشد، حال در **Error! Reference source not found.** به بررسی قطعه مرتبط بعدی onemesh_slave_key_offer() می‌پردازیم.

```
(...)
    print_debug("tdpOneMesh.c:2915","Enter..., rcvPkt->payload is %s",payload_dec);
    ret = tdp_onemesh_get_onemesh_info(onemesh_info);
    if (ret != 0) {
        print_debug("tdpOneMesh.c:2919","Failed tdp_onemesh_get_onemesh_info!");
        strncpy(error_msg,"Internal Error!",0xff);
    }
    parsed_obj = payload_parsing(payload_dec); if (parsed_obj == 0) {
        __s = "tdpOneMesh.c:2926";
        __dest = "Invalid rcvPkt";
        goto return_-1_n_exit;
    }
    str_obj = strcasestr_obj(parsed_obj,"method");
    if (((str_obj == 0) || (*(int *)(str_obj + 0xc) != 4)) || (str_obj = strcmp(*(char **)(str_obj + 0x10),"slave_key_offer"), str_obj != 0)) {
        __s = "tdpOneMesh.c:2934";
        __dest = "Invalid method!";
        goto return_-1_n_exit;
    }
    str_obj = strcasestr_obj(parsed_obj,"data");
    if ((str_obj == 0) || (*(int *)(str_obj + 0xc) != 6)) {
        __s = "tdpOneMesh.c:2941";
    }
    else {
        ret2 = strcasestr_obj(str_obj,"group_id");
        if (ret2 == 0) {
            __s = "tdpOneMesh.c:2948"; }
        else {
            __s = "tdpOneMesh.c:2948";
            if (*(int *)(ret2 + 0xc) == 4) {
                strncpy(onemesh_info_cp,onemesh_info,0x3f);
(...)
```

شکل ۶: بخش دوم onemesh_slave_key_offer()

در این قطعه توابع دیگری (توابع تنظیم شیء onemesh) برای تجزیه و تحلیل بار بسته فراخوانی می‌شوند. بار مورد انتظار، یک شیء JSON است، که در **Error! Reference source not found.** نمایش داده شده است.



```
{
    "method": "slave_key_offer",
    "data": {
        "group_id": "123",
        "ip": "1.3.3.7",
        "slave_mac": "00:11:22:33:44:55",
        "slave_private_account": "admin",
        "slave_private_password": "password",
        "want_to_join": false,
        "model": "owned",
        "product_type": "archer",
        "operation_mode": "whatever"
    }
}
```

شکل ۷: مثالی از بار JSON در onemesh_slave_key_offer()

قطعه بعدی نشان می‌دهد که هر کلید از شیء داده به ترتیب پردازش می‌شود. اگر یکی از کلیدها وجود نداشت، تابع خارج می‌شود. در **Error! Reference source not found.** این موضوع نمایش داده شده است.

```
(...)
    ret2 = strcasestr_obj(str_obj,"ip");
    if (ret2 == 0) {
        __s = "tdpOneMesh.c:2960";
    }
    else {
        __s = "tdpOneMesh.c:2960";
        if (*(int *)(ret2 + 0xc) == 4) {
            strncpy(slaveIp,*(char **)(ret2 + 0x10),0xf);
            print_debug("tdpOneMesh.c:2964","slaveIp is %s",slaveIp);
            ret2 = strcasestr_obj(str_obj,"slave_mac");
            if ((ret2 == 0) || (*(int *)(ret2 + 0xc) != 4)) {
                __s = "tdpOneMesh.c:2969";
            }
            else {
                strncpy(slaveMac_copy,*(char **)(ret2 + 0x10),0x11);
                strncpy(slaveMac,*(char **)(ret2 + 0x10),0x11);
                ret2 = strcasestr_obj(str_obj,"slave_private_account");
                if (ret2 == 0) {
                    __s = "tdpOneMesh.c:2978";
                }
                else {
                    __s = "tdpOneMesh.c:2978";
                    if (*(int *)(ret2 + 0xc) == 4) {
                        strncpy(encAccount,*(char **)(ret2 + 0x10),0x207);
                        ret2 = strcasestr_obj(str_obj,"slave_private_password");
                        if (ret2 == 0) {
                            __s = "tdpOneMesh.c:2986";
                        }
                        else {
                            __s = "tdpOneMesh.c:2986";
                            if (*(int *)(ret2 + 0xc) == 4) {
                                strncpy(encPassword,*(char **)(ret2 + 0x10),0x207);
                                ret2 = strcasestr_obj(str_obj,"want_to_join");
                                if (ret2 == 0) {
                                    __s = "tdpOneMesh.c:2995";
                                }
                                else {
                                    if (*(int *)(ret2 + 0xc) == 1) {
                                        ret2 = 1;
                                        acStack3190[0] = '1';
                                    }
                                    else {
                                        if (*(int *)(ret2 + 0xc) != 0) {
                                            __s = "tdpOneMesh.c:3013";
                                            goto ret_invalid_data;
                                        }
                                        ret2 = 0;
                                        acStack3190[0] = '0';
                                    }
                                    ret3 = strcasestr_obj(str_obj,"model");
                                    if ((ret3 == 0) || (*(int *)(ret3 + 0xc) != 4)) {
                                        __s = "tdpOneMesh.c:3021";
                                    }
                                    else {
                                        strncpy(model,*(char **)(ret3 + 0x10),0x20);
                                        ret3 = strcasestr_obj(str_obj,"product_type");
                                        if (ret3 == 0) {
                                            __s = "tdpOneMesh.c:3029";
                                        }
                                        else {
                                            __s = "tdpOneMesh.c:3029";
                                            if (*(int *)(ret3 + 0xc) == 4) {
                                                strncpy(prod_type,*(char **)(ret3 + 0x10),0x20);
                                                ret3 = strcasestr_obj(str_obj,"operation_mode");
                                                if (ret3 == 0) {
                                                    __s = "tdpOneMesh.c:3037";
                                                }
                                                else {
                                                    __s = "tdpOneMesh.c:3037";
                                                    if (*(int *)(ret3 + 0xc) == 4) {
                                                        strncpy(op_mode,*(char **)(ret3 + 0x10),0x10);
                                                        if (ret2 == 0) {
                                                            str_obj = create_csjon_obj(); if (str_obj == 0) {
                                                            __s = "tdpOneMesh.c:3132";
                                                            __dest = "Failed to creat cJSON Obj";
                                                            ret = -1;
                                                            print_debug(__s,__dest);
                                                            __s = NULL;
(...)
```

شکل ۸: بخش سوم onemesh_slave_key_offer()

همانطور که در شکل بالا مشاهده می‌شود، مقدار هر کلید JSON تجزیه شده و درون یک پشته متغیر کپی می‌شود (slaveMac، slaveIp و غیره). سپس تابع با فراخوانی create_csjon_obj() (0x405fe8) به بسته پاسخ می‌دهد.

از اینجا به بعد تابع، عملگرهای متغیری را روی داده‌های دریافتی اعمال می‌کند، بخش‌های مهم آن در **Error! Reference source not found.** نمایش داده شده‌اند.

```
(...)
    print_debug("tdpOneMesh.c:3363","Sync wifi for specified mac %s start.....",slaveMac);
    memset(systemCmd,0,0x200);
    snprintf(systemCmd,0x1ff,
        "lua -e \'require(\"luci.controller.admin.onemesh\"). \
        sync_wifi_specified({mac=\"%s\"})\'",slaveMac);
    print_debug("tdpOneMesh.c:3368","systemCmd: %s",systemCmd);
    system(systemCmd);
    print_debug("tdpOneMesh.c:3370","Sync wifi for specified mac %s end.....",slaveMac);
(...)
```

شکل ۹: بخش چهارم onemesh_slave_key_offer()

در **Error! Reference source not found.**، مقدار کلید JSON slave_mac، درون پشته متغیر slaveMac کپی شد. در **Error! Reference source not found.** slaveMac، توسط sprintf در متغیر systemCmd کپی شده و سپس به system() منتقل می‌شود.

۳ بهره‌برداری

۳-۱ دستیابی به تابع آسیب‌پذیر

اولین چیزی که باید مشخص شود چگونگی دستیابی به تزریق فرمان است. پس از آزمایش و خطا، محققان دریافتند ارسال ساختار JSON نمایش داده شده در شکل ۷ همیشه در مسیر کد آسیب‌پذیر قرار می‌گیرد. به طور خاص، این روش باید slave_key_offer باشد و استفاده از تابع want_to_join نادرست است. سایر مقادیر می‌توانند به طور دلخواه انتخاب شوند، اگرچه بعضی از کاراکترهای خاص در فیلدهایی غیر از slave_mac ممکن است باعث شوند که تابع آسیب‌پذیر زودتر از موعد خارج شود و تزریق را پردازش نکند.

با توجه به header بسته، همانطور که توضیح داده شد، مقدار نوع برابر 0xf0، opcode برابر ۶ و flagها برابر ۱ تنظیم می‌شوند تا فیلد checksum صحیح تنظیم شود.

۳-۲ رمزنگاری بسته

همانطور که در بخش گذشته توضیح داده شد، بسته با الگوریتم AES و کلید ثابت TPONEMESH_Kf!xn?gj6pMAt-wBNV_TDP رمزنگاری می‌شود. رمز در حالت CBC و IV دارای مقدار ثابت 1234567890abcdef1234567890abcdef است. علاوه بر این با وجود داشتن کلید ۲۵۶ بیتی و IV، الگوریتم واقعی مورد استفاده AES-CBC با کلید ۱۲۸ بیتی است، بنابراین نیمی از کلید و IV استفاده نمی‌شوند.

۳-۳ دستیابی به اجرای کد

اکنون می‌دانیم چگونه مسیر کد آسیب پذیر را بیابیم. برای ارسال بسته فرمان و اجرای کد، دو مشکل وجود دارد:

۱) تابع strncpy() تنها 0x11 بایت را از کلید slave_mac_info به متغیر slaveMac کپی می‌کند و بایت تهی را از بین می‌برد.
۲) از آنجایی که مقدار slaveMac به صورت تکی و دوتایی ذخیره می‌شود، باید داده‌ها را از آن خارج کرد.

با توجه به این دو محدودیت، فضای واقعی موجود کاملاً محدود است.

به منظور خارج کردن آرگومان‌ها و اجرای بار، باید کاراکترهای زیر را وارد کرد:

```
';<PAYLOAD>'
```

باید سه کاراکتر را از بین ببریم و تنها با ۱۳ بایت بار خود را بسازیم. با ۱۳ بایت (کاراکتر) اجرای هر کد معنی‌دار غیرممکن است.

به علاوه با انجام آزمایشاتی متوجه شدیم که حد در واقع ۱۲ بایت است. دلیل این مسئله مشخص نیست، اما به نظر می‌رسد مربوط به خروج داده است.

سیستم فایل‌های root خواندنی و نوشتنی است که یک اشتباه امنیتی بزرگ توسط TP-Link است. اگر این سیستم مانند اکثر دستگاههای تعبیه شده که از سیستم فایل‌های SquashFS استفاده می‌کنند، فقط خواندنی بود، این حمله خاص غیرممکن می‌شد؛ زیرا افزودن cd tmp ، بیش از ۱۲ کاراکتر موجود را مصرف خواهد کرد. حال باید دستورات را بایت به بایت ارسال کرده و آن‌ها را به یک پرونده فرمان «z» اضافه کنیم. سپس بار را ارسال کنیم:

```
sh z
```

حال پرونده فرمان به عنوان root اجرا می‌شود. از اینجا به بعد می‌توان دودویی را بارگیری و اجرا کرده و کنترل کامل روتر را در اختیار داشته باشیم.

۴ مراجع

[1] https://www.zerodayinitiative.com/blog/2020/4/6/exploiting-the-tp-link-archer-c7-at-pwn2own-tokyo