A report on the ASUS GT-AC5300 SQL Injection(Or even RCE in specified situation)

Vulnerability Product: GT-AC5300
Vulnerability Firmware Version: GT-AC5300_3.0.0.4_386_51569-g9ee6a79_ubi.w (latest)
Vulnerability type: SQL Injection(Or even RCE in specified situation)
Vulnerability Authentication Requirement: Low privilege

There is a SQL Injection in ASUS GT-AC5300. When an attacker has a low privilege account of the system, the attacker could trigger SQL injection through the `bwdpi_appStat` in `appGet.cgi` , and even cause RCE in specified situation

Contents

Reverse:

Firstly let's see the function of web api `bwdpi_appStat` in `/usr/sbin/httpd` , I call it `T_danger_SQL` here. (the following Figure 1)

you can find that here it received GET args from user, Then call `sqlite_Stat_hook` with the `client` , `mode` , `dura` , `date` received from the user as parameters(the following Figure 2), please attention that here is no any filter

In the following steps, you will find that the param `client` is not filtered until the end

```
000B4FCC                DCD aBwdpiRedirectI     ; "bwdpi_redirect_info"
000B4FD0                DCD sub_31B54
000B4FD4                DCD aBwdpiAppstat       ; "bwdpi_appStat"
000B4FD8                DCD T_danger_SQL
```

Figure 1

```
1int __fastcall T_danger_SQL(int a1, int a2)
2{
3  char *client; // r7
4  char *mode; // r4
5  char *dura; // r5
6  char *date; // r0
7  int v8; // [sp+10h] [bp-20h] BYREF
8
9  v8 = 0;
10  client = (char *)GET_POST("client");
11  if ( !client )
12    client = "";
13  mode = (char *)GET_POST("mode");
14  if ( !mode )
15    mode = "";
16  dura = (char *)GET_POST("dura");
17  if ( !dura )
18    dura = "";
19  date = (char *)GET_POST("date");
20  if ( !date )
21    date = "";
22  sqlite_Stat_hook(0, (int)client, (int)mode, (int)dura, (int)date, (int)&v8, a2);
23  return v8;
24}
```

Figure 2

Secondly, let's see the `sqlite_Stat_hook` in `/usr/lib/libbwdpi_sql.so` , you can find that the param `client` just was simply formatted into `where` without any further filter when `client != "all"` is true. (the following Figure 3)

then what happend to `where` ? You can see that when `mode == "hour" && dura == 24` is true, `where` is called as the third parameter to `j_bwdpi_appStat` (the following Figure 4)

```
  memset(having, 0, sizeof(having));
  memset(where, 0, sizeof(where));
  memset(s, 0, 0x1F40u);
  if ( !strcmp(client, "all") )
  {
    if ( strcmp(mode, "detail") )
    {
      v10 = "timestamp";
LABEL_8:
      strcpy(group, v10);
      goto LABEL_32;
    }
    if ( !a1 )
    {
      v10 = "mac";
      goto LABEL_8;
    }
    if ( a1 == 1 )
    {
      v10 = "app_name";
      goto LABEL_8;
    }
  }
  else
  {
    IF_mode_e_detail_then_0 = strcmp(mode, "detail");
    _IF_mode_e_detail_then_0 = IF_mode_e_detail_then_0;
    if ( IF_mode_e_detail_then_0 )
      _IF_mode_e_detail_then_0 = 1;
    if ( a1 )
      v13 = 0;
    else
      v13 = _IF_mode_e_detail_then_0 & 1;
    if ( v13 )
    {
      strcpy(group, "timestamp");
      snprintf(where, 0x40u, "app_name=\"%s\"", client);
      goto LABEL_32;
    }
    if ( ((unsigned __int8)_IF_mode_e_detail_then_0 & (a1 == 1)) != 0 )
    {
      strcpy(group, "timestamp");
```

Figure 3

```
LABEL_24:
      strcpy(group, "app_name");
      snprintf(where, 0x40u, "mac=\"%s\"", client);
      goto LABEL_32;
    }
  }
  puts("[sqlite] no such case!");
LABEL_32:
  if ( !strcmp(mode, "hour") )
  {
    if ( !strcmp((const char *)dura, "24") )
    {
      v16 = 1;
      v17 = fprintf(a7, "[");
      fflush(a7);
      v18 = *a6 + v17;
      v19 = v9 - 86370;
      *a6 = v18;
      do
      {
        memset(s, 0, 0x1F40u);
        v20 = v19;
        v19 += 3600;
        snprintf(having, 0x100u, "(SELECT * FROM traffic WHERE timestamp BETWEEN '%d' AND '%d')", v20, v19);
        j_bwdpi_appStat(s, group, where, having, 8000);
        if ( v16 )
          v21 = fprintf(a7, s);
        else
          v21 = fprintf(a7, ", %s", s);
        v22 = v21;
        fflush(a7);
        v23 = *a6 + v22;
        v16 = 0;
        *a6 = v23;
      }
```

Figure 4

Thirdly, let's see what happend in `j_bwdpi_appStat`. `j_bwdpi_appStat` call the `bwdpi_appStat` with the original parameters (the following Figure 5)

You can clearly find that when `having != NULL && where[0] != NULL` is true, the `where` is directly formatted into the `sql_query` and the SQL statement is executed without any filtering, resulting in SQL injection vulnerability (the following Figure 6)

```
// attributes: thunk
int __fastcall j_bwdpi_appStat(int s, int group, int where, int having, int a5)
{
  return bwdpi_appStat((char *)s, (const char *)group, (const char *)where, (const char *)having, a5);
}
```

Figure 5

```
  v43 = 0;
  memset(s, 0, sizeof(s));
  memset(sql_query, 0, sizeof(sql_query));
  memset(v47, 0, 0x12u);
  memset(v48, 0, sizeof(v48));
  memset(v52, 0, 0x1F40u);
  if ( *(_BYTE *)sub_1048("bwdpi_ana_path") )
    v8 = (const char *)sub_1048("bwdpi_ana_path");
  else
    v8 = "/jffs/.sys/TrafficAnalyzer/TrafficAnalyzer.db";
  v38 = file_lock("TrafficAnalyzer");
  if ( sqlite3_open(v8, &v43) )
  {
    v9 = (const char *)sqlite3_errmsg(v43);
    printf("Can't open database %s\n", v9);
LABEL_13:
    v10 = v43;
    goto LABEL_77;
  }
  strcpy(s, "mac, app_name, timestamp, SUM(tx), SUM(rx)");
  if ( !group )
    goto LABEL_13;
  if ( having )
  {
    if ( *where )
      snprintf(sql_query, 0x3C0u, "SELECT %s FROM %s WHERE %s GROUP BY %s", s, having, where, group);
    else
      snprintf(sql_query, 0x3C0u, "SELECT %s FROM %s GROUP BY %s", s, having, group);
  }
  else if ( !*where )
  {
    snprintf(sql_query, 0x3C0u, "SELECT %s FROM traffic GROUP BY %s", s, group);
  }
  if ( f_exists((int)"/tmp/BWSQL_LOG") > 0 )
  {
    snprintf(v51, 0x400u, "echo \"[BWDPI_SQLITE]sql_query = %s\" >> /tmp/BWSQL.log", sql_query);
    system(v51);
  }
  if ( !j_sql_get_table(v43, (int)sql_query, (int)&v46, (int)&v44, (int)&v45) )
  {
    if ( f_exists((int)"/tmp/BWSQL_LOG") > 0 )
```

Figure 6

More:

if `/tmp/BWSQL_LOG` exists, the `sql_query` will be formatted into command and executed directly, causing RCE (the following Figure 7)

```
{
  if ( *where )
    snprintf(v50, 0x3C0u, "SELECT %s FROM %s WHERE %s GROUP BY %s", s, a4, where, a2);
  else
    snprintf(v50, 0x3C0u, "SELECT %s FROM %s GROUP BY %s", s, a4, a2);
}
else if ( !*where )
{
  snprintf(v50, 0x3C0u, "SELECT %s FROM traffic GROUP BY %s", s, a2);
}
if ( f_exists((int)"/tmp/BWSQL_LOG") > 0 )
{
  snprintf(v51, 0x400u, "echo \"[BWDPI_SQLITE]sql_query = %s\" >> /tmp/BWSQL.log", v50);
  system(v51);
}
```

Figure 7

But it is difficult to make `/tmp/BWQL_LOG` exist, so no further discussion will be made here

HARM:

When an attacker has a low privilege account of the system, the attacker could trigger SQL injection through the `bwdpi_appStat` in `appGet.cgi` , attacker could SQL Inject by blind injection (such as time based injection, boolean based injection), and even cause RCE in specified situation

A Possible Poc:

```
/appGet.cgi?hook=bwdpi_appStat()&mode=hour&dura=24&client="OR"1"="1&date=1000
```

Please note that due to certain specific reasons (I currently do not have the device for further debugging), I am temporarily unable to attach the exp. If you do require an exp after replying to this email, I will resend a exp within a few weeks

Info

This vulnerability was submitted for research purposes, So I hope to obtain a CVE number for research purposes regardless of whether there is a bounty or not

discovered by leeya_bug
Security Researcher at NSFOCUS GeWu IoT Security Lab
Room604, building D, National Digital Publishing Base, No.996 Tiangu 7th Road, Yanta District, Xi'an