三个白帽之寻找来自星星的你 - 第二期解题分析

Via WooYun知识库 by 眼镜蛇

0x00 绕过auth验证

直接访问web服务器提示need cookie，通过IDA反汇编目标WEB服务程序， F5得到伪C代码如下：

```
haystack = strstr(byte_605240, "Cookie: ");
  if ( !haystack )
    the_exit(2, (__int64)"Need cookie", (__int64)&unk_4020E0, a1);
  qword_606248 = (__int64)strstr(byte_605240, "HTTP/1.0");
  if ( !qword_606248 )
  {
    qword_606248 = (__int64)strstr(byte_605240, "HTTP/1.1");
    if ( !qword_606248 )
      the_exit(2, (__int64)"Not supported version", (__int64)&unk_4020E0, a1);
  }
  if ( strncasecmp(byte_605240, "GET", 3uLL) )
    the_exit(2, (__int64)"Not supported method", (__int64)&unk_4020E0, a1);
  for ( i = 0LL; i < n; ++i )
  {
    if ( *(_BYTE *)(i + 6312512) == 13 || *(_BYTE *)(i + 6312512) == 10 )
      *(_BYTE *)(i + 6312512) = 0;
  }
  s = strstr(haystack, "auth=");
  if ( !s )
    the_exit(2, (__int64)"Need authentication", (__int64)&unk_4020E0, a1);
  sa = s + 5;
  if ( strlen(sa) > 0x40 )
    the_exit(2, (__int64)"Invaild authentication length", (__int64)&unk_4020E0, a1);
  sub_400F12(&var_60, sa);
  if ( memcmp(&var_60, s2, 0x20uLL) )
    the_exit(2, (__int64)"Authentication failed", (__int64)&unk_4020E0, a1);
```

通过分析得出WEB服务会从客户端发来请求的数据包里查找是否有Cookie关键字，然后再查找Cookie里是否有auth关键字，然后读取auth的值 再经过sub_400F12函数处理，把处理后的结果与 s2的前0x20个字节进行比较，如果相等 则验证成功，否则失败。

跟进400F12函数发现如下代码片段：

```
__int64 __fastcall sub_400F12(_BYTE *a1, _BYTE *a2)
{
  while ( v4 > 4 )
  {
    *v6 = 4 * byte_401FA0[(unsigned __int64)*v7] | ((unsigned __int8)byte_401FA0[(unsigned __int64)v7[1]] >> 4);
    v6[1] = 16 * byte_401FA0[(unsigned __int64)v7[1]] | ((unsigned __int8)byte_401FA0[(unsigned __int64)v7[2]] >> 2);
    v8 = v6 + 2;
    v6 += 3;
    *v8 = (byte_401FA0[(unsigned __int64)v7[2]] << 6) | byte_401FA0[(unsigned __int64)v7[3]];
    v7 += 4;
    v4 -= 4;
  }
  if ( v4 > 1 )
  {
    v9 = v6++;
    *v9 = 4 * byte_401FA0[(unsigned __int64)*v7] | ((unsigned __int8)byte_401FA0[(unsigned __int64)v7[1]] >> 4);
  }
  if ( v4 > 2 )
  {
    v10 = v6++;
    *v10 = 16 * byte_401FA0[(unsigned __int64)v7[1]] | ((unsigned __int8)byte_401FA0[(unsigned __int64)v7[2]] >> 2);
  }
  if ( v4 > 3 )
  {
    v11 = v6++;
    *v11 = (byte_401FA0[(unsigned __int64)v7[2]] << 6) | byte_401FA0[(unsigned __int64)v7[3]];
  }
  *v6 = 0;
  return v13 - (-v4 & 3u);
}
```

是很明显的BASE64解密算法代码

接下来看看s2的内容是什么。这段代码里发现

```
sub_4014D3(int a1, unsigned int a2, void *a3)
s2 = a3;
```

那么s2是由外部变量a3传递进来的，我们返回到主函数main里追踪一下a3变量的来历。在主函数里有以下代码片段：

```
sub_401125("../server.cfg", &v12);
sub_4014D3(v10, i, &v12);
```

大概是先通过函数401125得到v12,然后传递给4014D3函数的，这里v12对应上面分析的a3。再看看401125函数的功能：

```
int __fastcall sub_401125(const char *a1, char *a2)
{
  FILE *stream; // [sp+20h] [bp-10h]@1
  char *v4; // [sp+28h] [bp-8h]@1

  stream = fopen(a1, "r");
  fgets(a2, 32, stream);
  v4 = strpbrk(a2, "\r\n");
  if ( v4 )
    *v4 = 0;
  return fclose(stream);
}
```

很明显就是直接读取文件 ../server.cfg的前32个字节的内容到v12里。

到这里，我们就明白了，实际上auth就是文件../server.cfg 前32个字节base64加密后的结果。

可是我们怎么来得到这32个字节的内容呢？

通过仔细阅读伪C代码发现，打印错误信息函数the_exit有如下代码片段：

```
else if ( a1 == 2 )
  {
    sub_401195(v10, 0x194u, (__int64)&unk_4020E0);
    sprintf(s, "<HTML><BODY><H1>WebServer: %s %s</H1></BODY></HTML>", a2, v9);
    v6 = strlen(s);
    sub_4012E7(v10, s, v6);
    sprintf(s, "SORRY: %s-%s", a2, v9);
  }
```

跟进401195 发现：

```
if ( qword_606248 )
    v3 = (const char *)qword_606248;
  else
    v3 = "HTTP/1.0";
  v4 = snprintf(&s, 0x100uLL, "%s %d %s\r\n", v3, a2, a3, a3, __PAIR__(a1, a2));
  result = write(fd, &s, v4);
  v7 = *MK_FP(__FS__, 40LL) ^ v9;
  return result;
```

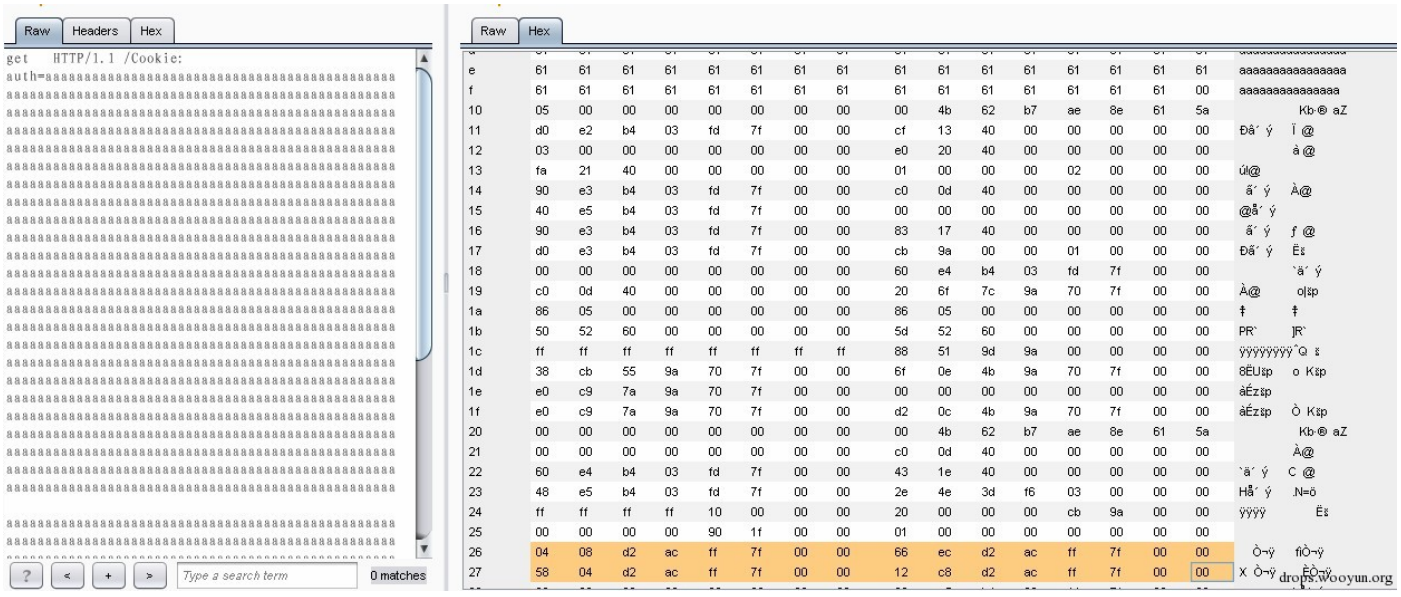
这里snprintf函数将格式化长度为0x100个字节的数据到s，其返回值v4是字符串实际的长度值，也就是说完全是可以大于0x100这个长度的。

接着Write函数将格式化后的字符串 取长度v4返回给客户端，../server.cfg的前32个字节是在主函数里读取的，这里如果我们控制了长度v4完全可能造成内存数据泄露，从而泄露../server.cfg的前32个字节的值，这样就有可能得到auth。

我们先尝试构造超长HTTP数据包试试 访问个超长名字的目录结果如下：

提示找不到cookie，那么webserver是通过函数strstr来查找cookie的 那么cookie放到任何位置都不影响。

我们把cookie放到第一行再尝试：

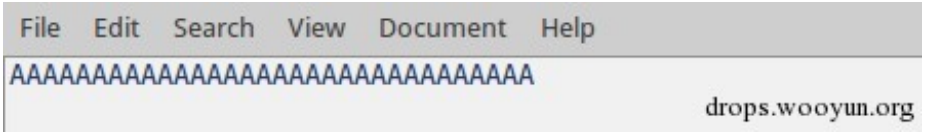


成功越界访问内存。那么哪部分才是auth呢？

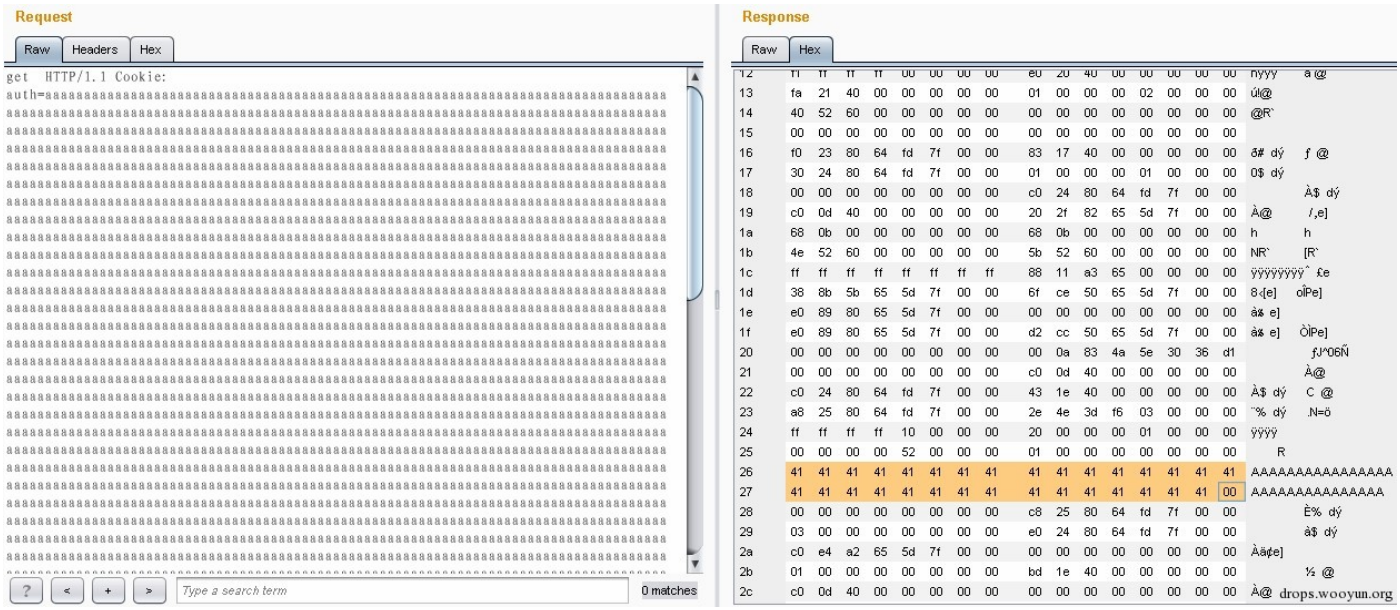
这里我们本地测试一下：

在本地运行webserver 然后来一次内存泄露，看看auth在哪个位置

在本地构造server.cfg内容为32个A，如图：



运行服务，然后发送超长HTTP数据包如图：



在0x26偏移处看到了auth，那么对应三个白帽服务器的auth 应该也在0x26处。

将该处32字节进行base64编码得到

auth=BAjSrP9/AABm7NKs/38AAFgE0qz/fwAAEsjSrP9/AAA=

0x01 绕过目录跳转符号..检查

通过auth验证后，有如下代码：

```
file = (char *)&unk_605244;
  v8 = strlen((const char *)&unk_605244);
  v15 = 0LL;
  while ( v8 > v15 && isspace(file[v15]) ) //去掉目录名前面的空格
  {
    ++v15;
    ++file;
  }
  for ( j = 0LL; ; ++j )
  {
    if ( strlen(file) <= j )
      goto LABEL_38;
    if ( isspace(file[j]) )
      break;
  }
  file[j] = 0;   //FILE目录名里遇到空格便用\x0截断
LABEL_38:
  v9 = strlen(file);
  for ( k = 0; k < v9; ++k )
  {
    if ( file[k] == 46 && file[k + 1] == 46 ) //如果目录里出现连续的.. 就提示错误
      the_exit(2, (__int64)"Path travel not supported", (__int64)file, a1);
  }
```

通过分析，除非能控制file的长度让函数还没检查到..时就停止检查。

File长度v9 =strlen(file), 能控制srlen函数返回值的 就是字符0x00了，也就是file里要出现一个字符0x00，但是还要保证能正常读取文件，文件还必须存在。

这时我们看看后面读取文件的代码

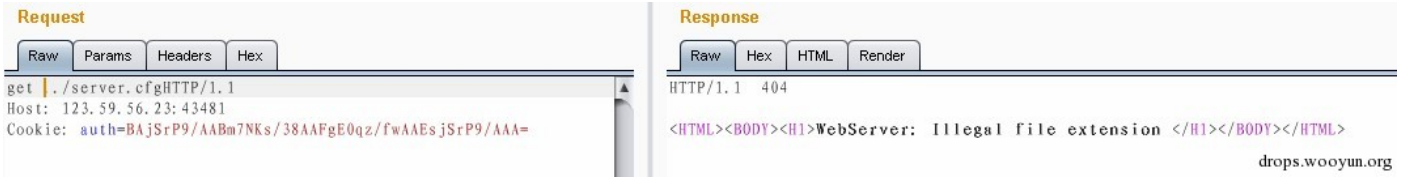
```
filea = file + 1;
v10 = open(filea, 0, s2);
```

读取文件时读取的是filea，如果我们让file[0]=0x0那么就能绕过..检查了。怎么来制造一个\x0字符呢？

在函数前部分有如下代码片段：

```
for ( i = 0LL; i < n; ++i )
  {
    if ( *(_BYTE *)(i + 6312512) == 13 || *(_BYTE *)(i + 6312512) == 10 )
      *(_BYTE *)(i + 6312512) = 0;
}
```

它的作用是http数据包里把所有的换行符号0x0d,0x0a替换成0x00，如果我们把让file[0]=0x0d，那么在检查.. 时 file[0]就变成了，0x00,他后面跟../server.cfg后面再跟一个0x0d截断，这样就可以绕过目录跳转检查了，如图：



0x02 绕过文件后缀检查

文件后缀检查有如下代码片段：

```
v11 = 0LL;
  for ( l = 0LL; ; ++l )
  {
    if ( !off_603160[2 * l] )
      goto LABEL_49;
    v3 = (signed int)strlen(off_603160[2 * l]);
    v4 = off_603160[2 * l];
    v5 = strlen(file);
    if ( !strncmp(&file[v5 - v3], v4, v3) )
      break;
  }
  v11 = off_603160[2 * l + 1];
LABEL_49:
  if ( !v11 )
    the_exit(2, (__int64)"Illegal file extension", (__int64)&unk_4020E0, a1);
}
```

代码功能就是从数组603160里读后缀白名单，如果提交的文件后缀在白名单里 就通过，否则就不通过。我们看下603160的文件后缀白名单内容如下：

```
a:0000000000401F20 unk_401F20       db  67h ; g
a:0000000000401F21                  db  69h ; i
a:0000000000401F22                  db  66h ; f
a:0000000000401F23                  db    0
a:0000000000401F24 aImageGif        db 'image/gif',0
a:0000000000401F2E aJpg             db 'jpg',0
a:0000000000401F32 aImageJpeg       db 'image/jpeg',0
a:0000000000401F3D aPng             db 'png',0
a:0000000000401F41 aImagePng        db 'image/png',0
a:0000000000401F4B aHtm             db 'htm',0
a:0000000000401F4F aTextHtml        db 'text/html',0
a:0000000000401F59 aXml             db 'xml',0
a:0000000000401F5D aTextXml         db 'text/xml',0
a:0000000000401F66 aTz              db 'tz',0
a:0000000000401F69 aImageGz         db 'image/gz',0
a:0000000000401F72 aJs              db 'js',0
a:0000000000401F75 aTextJs          db 'text/js',0
a:0000000000401F7D aCss             db 'css',0
a:0000000000401F81 aTextCss         db 'text/css',0
```


由于前面我们让file[0]=0x0d 准换后就是0x00所以这里v5 = strlen(file); v5-v3=-v3，file指针是个负数，因此file前面-v3开始的字符串要存在文件后缀白名单里才能通过验证。File指针指向的内存前后内容如下：

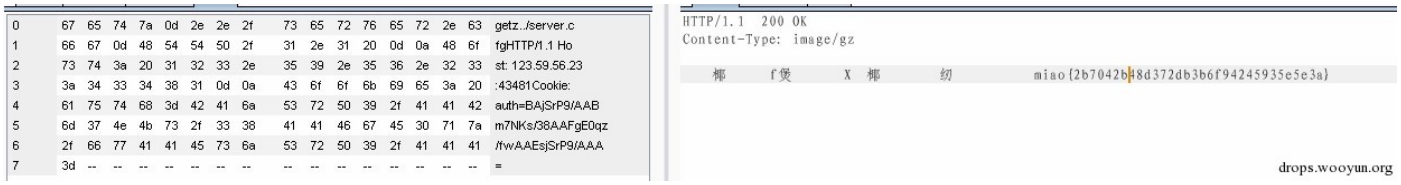
get [0x0d]../server.cfg[0x0d]

file指向第一个0x0d，它前面get是不能控制的，否则http数据包就无法成功被识别，get检查片段：

```
if ( strncasecmp(byte_605240, "GET", 3uLL) )
    the_exit(2, (__int64)"Not supported method", (__int64)&unk_4020E0, a1);
```

能控制的就是t后面那个空格字符，所以我们就要考虑一下后缀有没有get et t开头的字符串了，如果有的话，就尝试构造这样的后缀名。

观察发现只有tz是以t开头的，满足条件。把get后面的空格替换成z 刚好满足条件。于是成功读取到server.cfg的内容，如图：



从server.cfg里成功得到了flag！